BEST PRACTICES FOR MIGRATING TERADATA TO GOOGLE CLOUD PLATFORM

Ekaba Bisong, *Data Developer/ Scientist, Google Cloud Data Engineer*
Gurinderbeer Singh, *Data Developer*
John Schulz, *Principal Consultant*
Kartick Sekar, *Solutions Architect*
Paul Melanson, *Big Data Team Manager*
Scott McCormick, *Google Cloud Certified Solution Architect*
Vladimir Stoyak, *Principal Consultant for Big Data, Google Cloud Platform Qualified Solution Developer*

This white paper documents the process of migrating data from Teradata to Google Cloud Platform. It highlights several key areas to consider when planning a migration of this nature, including pre-migration considerations, details of the migration phase, and best practices.

This white paper takes a non-technical approach to process description and provides example data pipeline architectures.

MOTIVATORS TO MIGRATE

Google BigQuery is a fully managed cloud enterprise data warehouse. It leverages the power of Google's technologically advanced storage and processing infrastructure to provide an extremely fast, scalable, serverless, no-operations (No-Ops) database solution to clients. This managed platform abstracts clients from the overhead burden of storing and processing large datasets.

One key attraction of BigQuery as a preferred platform for enterprise data warehousing is the reduction of operations, to treat the database as serverless. Going serverless and No-Ops means that clients can immediately start using BigQuery as a storage solution without bothering about configuring disks, warehouse security, high availability, memory updates, load balancing, and so on.

Google BigQuery is a petabyte-scale low-cost enterprise data warehouse for analytics. Customers find BigQuery's performance and ease of use liberating, allowing them to experiment with enormous datasets without compromise and to build complex analytics applications such as reporting and data warehousing.

Some of the key reasons enterprises migrate off of Teradata:

- Technology fails to meet the needs of today's business users, such as the increasing requirement for unlimited concurrency and performance.
- Key data sources for the modern enterprise are already in the cloud. The cloud also allows for new types of analytics to be assessed and refined without a long-term commitment to infrastructure or specific tools.
- Pay-as-you-go cloud scalability without the need for complex reconfiguration as your data or workloads grow.

Here are some of the main reasons that users find migrating to BigQuery an attractive option.

- BigQuery is a fully managed, NoOps data warehouse. In contrast to Hadoop systems, the concept of nodes and networking are completely abstracted away from the user.
- BigQuery enables extremely fast analytics on a petabyte scale through its unique architecture and capabilities.
- BigQuery eliminates the need to forecast and provision storage and compute resources in advance. All the resources are allocated dynamically based on usage.

- BigQuery provides a unique pay-as-you-go' model for your data warehouse and allows you to move away from a CAPEX-based model.
- BigQuery charges separately for data storage and query processing, enabling an optimal cost model—unlike solutions where processing capacity is allocated (and charged) as a function of allocated storage.
- BigQuery employs a columnar data store, which enables the highest data compression and minimizes data scanning in common data warehouse deployments.
- BigQuery provides support for streaming data ingestions directly through an API or by using Google Cloud Dataflow.
- BigQuery provides the ability to connect to federated (external) data sources such as Google Cloud Bigtable, Google Cloud Storage (GCS) and Google Drive.
- BigQuery has native integrations with many third-party reporting and BI providers, such as Tableau, MicroStrategy and Looker.

Here are some scenarios where BigQuery might not be a good fit.

- BigQuery is not an OLTP database. It performs full column scans for all columns in the context of the query. It can be very expensive to perform a single row read similar to primary key access in relational databases using BigQuery.
- BigQuery was not built to be a transactional store. If you are looking to implement locking, multi-table transactions, BigQuery is not the right platform.
- BigQuery does not support primary keys and referential integrity.
- BigQuery is a massively scalable distributed analytics engine. Using it for querying smaller datasets is overkill because you cannot take full advantage of its highly distributed computational and I/O resources.

PERFORMANCE AND USAGE

In an analytical context, performance has a direct effect on productivity because running a query for hours means days of iterations between business questions. It is important to know that a solution will scale well not only in terms of data volume but in the quantity and complexity of the queries performed.

For queries, BigQuery uses the concept of "slots" to allocate query resources during query execution. A slot is simply a unit of analytical computation (i.e., a chunk of infrastructure) pertaining to a certain amount of CPU and RAM. All projects, by default, are allocated 2,000 slots on a best-effort basis. As the use of BigQuery and the consumption of the service goes up, the allocation limit is dynamically raised.

DENORMALIZING DATA

It's important to recognize that BigQuery uses a fundamentally different architecture than Teradata, and a traditional star or snowflake schema may not perform as well as could be expected. BigQuery offers the opportunity to store data from different tables in the same table, speeding up data access considerably.

The preferred method for denormalizing data takes advantage of BigQuery's native support for nested and repeated structures in JSON or Avro input data. Expressing records using nested and repeated structures can provide a more natural representation of the underlying data. In the case of the sales order, the outer part of a JSON structure contains the order and customer information, and the inner part of the structure contains the individual line items of the order, which are represented as nested, repeated elements.

MULTISET TABLES

Teradata multiset tables allow having duplicate rows unless explicit unique indexes are defined on the table. In the latter case, the table will behave like an ordinary set table. Multiset tables are mostly used on landing layers.

Multiset tables without unique indexes map directly to BigQuery tables. BigQuery has no primary or unique key and no index concept. So, duplicate rows are allowed. For such tables, only the most recent row per identifying column combination is returned either in the direct query or through the so-called watermark view.

SET TABLES

In Teradata, set tables do not allow duplicate rows. Duplicates might just not be added or an exception could be thrown depending on how the data was inserted. In BigQuery there is no direct equivalent for such tables. Pythian recommends data from all tables be deduplicated on fetch.

GLOBAL TEMPORARY TABLES

There is no concept of temporary tables in BigQuery. An alternative in BigQuery is using a table per request with defined expiration time. After the table is expired it is removed automatically from the BigQuery storage. The minimal expiration time for a table is 3600 seconds.

PARTITIONING

BigQuery supports partitioning on a single column of one of the data types.

WILDCARDS

Querying multiple tables in BigQuery at once is also possible by using wildcard queries. Additionally, the view can be added to run a wildcard query against all tables in the group. If there is a need to query data for a single customer, a specific table name can be specified. Though wildcard queries are less performant than querying a partitioned table, this is a good way to control the volume of scanned data.

INFRASTRUCTURE, LICENSE AND MAINTENANCE COSTS

With BigQuery, Google provides a fully managed serverless platform that eliminates hardware, software and maintenance costs. BigQuery also reduces efforts spent on detailed capacity planning, thanks to embedded scalability and a utility billing model: services are billed based on the volume of data stored and amount of data processed.

Here are the important specifics of BigQuery billing.

- Storage is billed separately from processing, so working with large amounts of “cold” (rarely accessed) data is very affordable (price for data not edited for >90 days drops by 50 percent).
- Automatic pricing tiers for storage (short term and long term) eliminate the need for commitment to and planning for volume discounts; they are automatic.
- Processing cost is only for data scanned, so selecting the minimum number of columns required to produce results will minimize the processing costs, thanks to columnar storage. This is why explicitly selecting the required columns instead of “*” can make a big processing cost (and performance) difference.
- Likewise, implementing appropriate partitioning techniques in BigQuery will result in fewer data scanned and lower processing costs (in addition to better performance). This is especially important because BigQuery doesn’t utilize the concept of indexes that are common to relational databases.
- There is an extra charge if a streaming API is used to ingest data into BigQuery in real time.
- Google provides a “fast-access” storage API for a higher cost. When you use the BigQuery Storage API, structured data is sent over the wire in a binary serialization format. This allows for additional parallelism among multiple consumers for a set of results.
- Cost-control tools such as “Billing Alerts” and “Custom Quotas” and alerting or limiting resource usage per project or user might be useful.
- Google offers a flat-rate pricing for customers who prefer a fixed-billing model for data processing. Although usage-based billing might be a better option for most of the use cases and clients, receiving a predictable bill at the end of the month is attractive to some organizations.

REPORTING, ANALYTICS AND BI TOOLSET

Reporting, visualization and BI platform investments, in general, are significant. Although some improvements might be needed to avoid bringing additional dependencies and risks into the migration project (BI/ETL tools replacement), keeping existing end-user tools is a common preference.

There is a growing list of vendors that provide a native connection to BigQuery (https://cloud.google.com/bigquery/partners/). However, if native support is not yet available for the BI layer in use, Google has partnered with Simba Technologies to provide ODBC and JDBC drivers.

Many clients are often stuck with an older reporting platform that does not allow them to take advantage of BigQuery's architecture and optimizations. In those cases, they often decide to move to a new platform quickly after the BigQuery migration. This gives our clients the chance to take advantage of the many best practices in reporting such as:

1. Multiple, smaller tools for specific jobs
2. Robust security models within the reporting platform or BigQuery
3. Cost optimization strategies
4. Centralized dimension datasets
5. Templating for rapid development

Traditional approaches for reporting and analytics often push the concept that one tool should service all reporting requests. This can be a very expensive proposition, and we recommend that BI functionality not be limited to a single tool, if possible. Even if current use cases point to a single tool, it is still prudent to ensure that any reporting/analytical solution refrains from embedding too much business logic in the tool itself. Although many reporting environments provide a rich set of convenience functions, utilizing these functions to provision common business logic will mean restricted reuse and a requirement to duplicate the logic wherever the original tool can't be used.

Security also plays a major factor. BigQuery currently does not support row level or column level security at a table level. To facilitate this kind of security, a robust BI tool can provide the desired controls. There are two alternatives for implementing security within BigQuery. The first is to control user access at a dataset level using ACLs and IAM Policies. The second is to implement data control structures joined in for every query at the database level. This approach can be quite flexible, but can also be a very complex and administratively intense undertaking.

Cost concerns also drive optimizations to reduce cost. In some instances, the BI tool may support a caching mechanism to reduce the amount of CPU needed from BigQuery. It is important to understand the various options available at the BigQuery level and at the BI tool level. In addition, BI tools are not always efficient in generating optimized queries and continuous analysis of long-running or high volume queries is recommended. Tuning of these types of queries can make a significant difference in managing cost; especially if these queries are frequently executed.

Of all the dimensions that may exist in the underlying data model, the time dimension is easily one of the most critical. Although many BI tools provide convenience functions for time transformations, we recommend that these transformations be persisted at the database level. This will provide a convenience layer for persisting time transformed aggregates into summary/aggregate tables.

Lastly, when evaluating a BI tool, it is important to understand the degree of abstraction the tool can provide. This will enable strong inheritance rules and template driven report development. Inheritance rules will provide a consistent means of enforcing common definitions across calculations. Template support is a key capability for rapid report development and if self-serve offerings are to be considered.

USABILITY AND FUNCTIONALITY

There may be some functional differences when migrating to a different platform. However, in many cases, there are workarounds and specific design considerations that can be adopted to address these.

BigQuery launched support for standard SQL, which is compliant with the SQL 2011 standard and has extensions that support querying nested and repeated data. The query editor allows toggling between standard SQL and legacy SQL. Standard SQL is more similar to conventional SQL, and this option is recommended.

It is worth exploring some additional SQL language/engine specifics and some additional functions:

- analytic/window functions
- JSON parsing
- working with arrays
- correlated subqueries
- temporary SQL/Javascript user-defined functions
- inequality predicates in joins
- table name wildcards and table suffixes

In addition, BigQuery supports querying data directly from GCS and Google Drive. It supports AVRO, JSON NL, CSV files as well as Google Cloud Datastore backup and Google Sheets (first tab only).

Federated data sources should be considered in the following cases:

- Loading and cleaning your data in one pass by querying the data from a federated data source (a location external to BigQuery) and writing the cleaned result into BigQuery storage.

- Having a small amount of frequently changing data that you join with other tables. As a federated data source, the frequently changing data does not need to be reloaded every time it is updated.

As a guideline, measurable metrics in at least the following three categories need to be established and observed throughout the migration process: performance, infrastructure/license/maintenance costs, and usability/functionality.

DATA MIGRATION

Typically for the initial load process, we export data from the source system into a flat file and then loading it into BigQuery. GCP provides a service called the BigQuery Data Transfer Service which has the connectors and tooling to export the data out of your Teradata instance onto GCS and then subsequently load it into BigQuery. Find more information on this service here. Please note that there are some data type differences between Teradata and BigQuery. Generally, the most defensive data type is used by default to be loaded on to BigQuery. In some cases where data types cannot be directly inferred or there is a need to preserve to a compatible type it may be necessary to do some additional massaging to the data in BigQuery.

Typical data migration strategies include a few steps:

1. Identify the base data set and point in time data (TD Queries) for the initial load.
2. Use the data transfer service or an equivalent tool/process to extract the base data to GCS and then import to BigQuery.
3. Identify process/queries for delta data (while the rest of the migration process is ongoing) and Identify load of delta data to BigQuery.
4. Identify Data concurrency and consistency test cases.
5. Create the process for loading delta data from TD (via the extract to GCS) process to BigQuery.
6. Execute the data concurrent & consistency test cases.

DATA TRANSFORMATIONS & INTEGRATION

Any data migration endeavor is incomplete without also moving the tooling that processes, transforms and loads the data into the warehouse. In Teradata, this implies the BTEQ and FLD scripts that are used for data loading and massaging.

Typical EDW processes use either an ETL (Extract, Transform & Load) or an ELT (Extract, Load and Transform) strategy for data massaging. Both ETL & ELT strategies can be used very effectively with BigQuery as the primary & final destination data store and GCS as the staging (or ingest) or intermediate data store as required.

Here are some methods that we have used for transformations:
For most batch based scenarios:

Extract:
Consider landing the source data in GCS in either JSON, AVRO, CSV or PARQUET formats (there are other formats that are also supported). There are pros/cons to using each of these formats for initial data export and subsequent loading to BigQuery. For a deeper understanding of what format works best for your use case, find more information here.

Transform & Load (or vice-versa):
Data transformation can happen with a number of tools available natively within the GCP ecosystem as managed services. Some of the key tools available are:

Cloud Dataflow: Cloud Dataflow is a fully-managed service for transforming and enriching data both in stream (real time) and batch (historical) modes. With its serverless & fully managed approach to resource provisioning and management, one can focus on the core transformational logic without worrying about capacity, resources & scaling & underlying infrastructure.

Cloud Dataproc: Cloud Dataproc is a fully managed service for running Apache Spark & Hadoop. Cloud Dataproc provides the management and capabilities for running your transformations at scale with very efficient, fully managed and rapid provisioning of clusters to run your jobs.

A few key points to consider when thinking about using Dataflow Vs Dataproc in terms of their key capabilities:

Capability	Dataflow	Dataproc
Streaming & NRT	✓	X
Batch	✓	✓
Iterative Processing & Notebooks	X	✓
Spark + Spark ML	X	✓

There are additional tools that enable the transformation process such as:

Cloud Composer: is a fully managed google implementation of Apache Airflow that allows for orchestration, choreography for workflows and allows for authoring, scheduling and monitoring these workflows.

Cloud Data Fusion: Cloud Data Fusion is a fully managed WYSIWYG GUI based interface built on the open source CDAP project which allows for cloud-native data integration without the need for writing extensive code. This has a rich ecosystem of pre-built connectors and transformations and allows organizations to rapidly develop and deploy data pipelines in the cloud.

Talking specifically about Load: BigQuery allows for a direct load of the data into staging table from GCS (depending on the source data formats). In some cases, the transformations for the data may be simple enough to ingest the source data as is and create "views"

LOGGING MONITORING & AUDITING

Early-stage provisioning of logging processes and performance monitoring is recommended to ensure the long-term success of a Google Cloud Platform project. Stackdriver provides wide-ranging monitoring of Google Cloud Platform resources to grant awareness and visibility across the platform.

STACKDRIVER CONFIGURATION

Accounts will hold the monitoring configurations for a group of Google Cloud Platform projects. Each GCP project can only be associated with one Stackdriver account. One option is to set up one Stackdriver account per GCP project. The other option is to have a central Stackdriver account monitoring multiple GCP projects. Within one Stackdriver account, users would have access to the same dashboard and other resources.

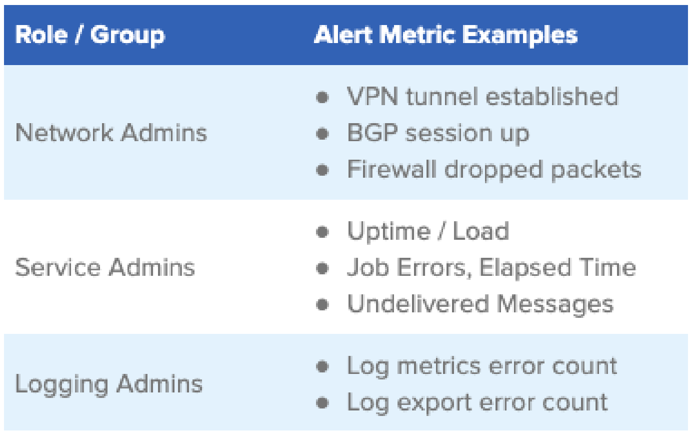
Pythian uses a Stackdriver account for each environment within its own project. The Stackdriver account will monitor both the running environment and its corresponding control plane. Each project exports logs via an export sink to the project folder. We also export to BigQuery for log auditing.

Every time a log entry arrives in a project, folder, billing account, or organization resource, Stackdriver Logging compares the log entry to the sinks in that resource. Each sink whose filter matches the log entry writes a copy of the log entry to the sink's destination. Since exporting happens for new log entries only, you cannot export log entries that Stackdriver Logging received before your sink was created.

MONITORING AND ALERTS

Stackdriver agents gather system and application metrics from virtual machine instances and send them to Stackdriver monitoring. By default, the agent collects disk, CPU, network, and process metrics. You can configure the agent to monitor third-party applications as well.

The following is a list of alerts that customers might be interested in setting up. However, there are many additional metrics which can be used for alerting. The full list per service can be found here.



Role / Group	Alert Metric Examples
Network Admins	• VPN tunnel established • BGP session up • Firewall dropped packets
Service Admins	• Uptime / Load • Job Errors, Elapsed Time • Undelivered Messages
Logging Admins	• Log metrics error count • Log export error count

LOGGING

Stackdriver Logging allows you to store, search, analyze, monitor and alert on log data and events from GCP. The API also allows you to ingest any custom log data from any source. Analyze high-volume application and system logs in real-time. Logs are associated with GCP projects, however, the organization can also have logs. The Logs Viewer only shows you log data for one GCP project at a time, however, with the API you can review log entries for multiple projects at one time.

We leverage a combination of Stackdriver, GCS, and BigQuery Logs are processed through Stackdriver and exported into GCS or processed by BigQuery.

Also See:
StackDriver Export Design Patterns
StackDriver Export

LOGS ACCESS

Access control provides flexible and effective tools that you can use to protect sensitive data. In Google Cloud Platform access to logs are controlled at the project level by IAM role(s) that are given to a user or service account. Roles can include Stackdriver Logging IAM roles as well as legacy project roles which control access to project resources including logging. Without any Stackdriver Logging or project role, you cannot view the information in the Logs Viewer and cannot use the Stackdriver Logging API or a command-line interface to access logging information. IAM Logging roles include Logging permissions and can be assigned to users, groups, and service accounts that belong to a project or other resource that can contain logs.

Google recommends creating monitoring groups that will be assigned IAM logging roles. Apply the same policies which exist today for accessing various log types. Define which user groups should have specific IAM roles for logging and apply them in the Stackdriver project as well as the project-level of the project(s) being monitored by Stackdriver. IAM logging roles can be found here.

AUDIT CONTROL

Within Stackdriver, Cloud Audit Logging maintains three audit logs for each project, folder, and organization: Admin Activity, Data Access, and System Event. Audit log entries are written to these logs to answer the questions of "who did what, where, and when?"

Admin Activity audit logs contain log entries for API calls or other administrative actions that modify the configuration or metadata of resources. For example, the logs record when users create VM instances or change Cloud Identity and Access Management permissions.

Data Access audit logs record user-driven API calls that create, modify, or read user-provided data. Please note this is data access outside of BigQuery for objects such as GCS Buckets or Compute Engine.

System Event audit logs contain log entries for GCP administrative actions that modify the configuration of resources. System Event audit logs are generated by Google systems; they are not driven by direct user action.

Every audit log entry in Stackdriver Logging is an object of type LogEntry that is characterized by the following information:

- The project or organization that owns the log entry.
- The resource to which the log entry applies. This consists of a resource type from the Monitored Resource List and additional values that denote a specific instance.
- A log name.
- A timestamp.
- A payload, which is the protoPayload type. The payload of each audit log entry is an object of type AuditLog, a protocol buffer, and contains a field, serviceData, that some services use to hold additional information.

Outside of Stackdriver, BigQuery supports two versions of log messages: AuditData (the older version) and BigQueryAuditMetadata (the newer version). The older AuditData log entries tend to map directly to individual API calls made against the BigQuery service.

The newer format, BigQueryAuditMetadata, represents a better view into BigQuery operations. BigQueryAuditMetadata provides details on how resources are changed indirectly by other resources, particularly when reporting on asynchronous events and events that are not strongly coupled to a particular API call. For example, the BigQueryAuditMetadata entries can report when BigQuery tables are removed because of a configured expiration time. This is not possible with the older logging format.

The BigQuery logs are automatically sent to Stackdriver for reporting and filtering.

DATA QUALITY/VALIDATION

We need to confirm whether our data was moved successfully. Finding a tool to compare and make simple validations between environments is challenging.

We developed a framework for comparing datasets between environments using a YAML-based configuration. We started with a few simple requirements.

1. Tests should be easily configurable using a YAML-based file structure
2. Tests should be grouped together so they can be run at certain times. For example, after moving table1, table2 and table3, we'd want to run a test proving that tables 1, 2 and 3 in the old and new environments matched.
3. We should have the ability to cache the test results for a dashboard but force a refresh when necessary.
4. Any data transformation rules should be reflected in/tested by the QA process.
5. We should be able to run simple counts tests as well as some ad-hoc queries.

We then created a simple test framework.
An example test in the yaml configuration looks like this:

data_qa.yaml:

```
data_qa_group_01:
    tests:
         straight_counts_part_02:
            Type: SimpleCount
            Env_A: teradata_dev# Mapping in db_creds.py
    Env_B: bigquery_dev # Mapping in db_creds.py
            # List of tables in Environment A
            Tables_A: [customer, member, sales]
            # List of tables in Environment B
            Tables_B: [demo.customer, demo.member, demo.
sales]
            Assert: equal # Assert that counts are equal
            Notes: Refer to ticket ABC123
```

db_creds.py:

```
Environments = {
      'teradata_dev' : {
```

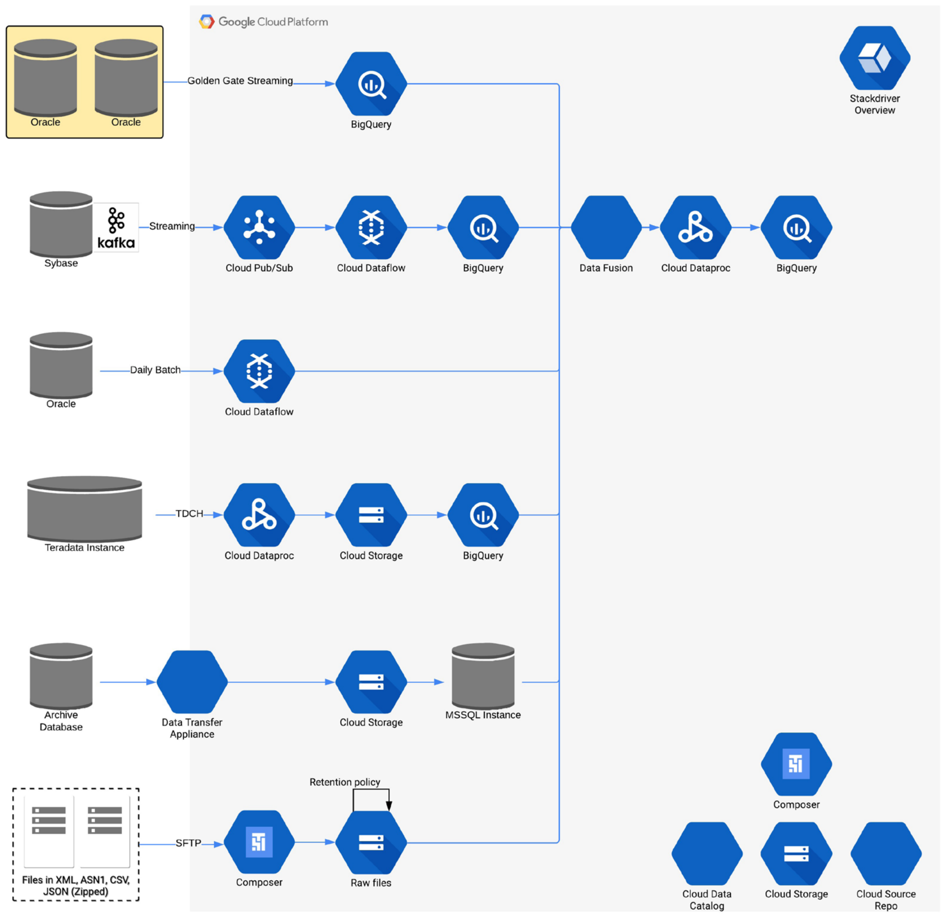
```
                    'connection_type':'teradata',
                   'jclassname':'com.teradata.jdbc.TeraDriver',
                   'user': 'td_user',
                   'password': 'pass+word',
                   'host': '127.0.0.1',
                   'database': 'dw_dev',
                   'raise_on_warnings': False
               },
       'bigquery_dev' : {
                   'connection_type':'bigquery',
                   'project_id':'my-bq_proj'
               }
                               }
```

Running a test to compare these three tables would be as simple as running the following command (note that force_refresh is optional):

```
python data_qa.py --test_group data_qa_group_01 --force_
refresh True
```

MIGRATION APPROACH OPTIONS

The below diagram is a high-level overview of multiple source systems and a Teradata instance that Pythian migrated to BigQuery.



The source systems consisted of four active OLTP databases of either Oracle or Sybase, a database of archive data which was not being changed, large sets of flat files in various formats, and the Teradata instance itself.

For each of these systems, we needed to develop a means of ingesting the data, a pipeline architecture to process and transform the data, and a combined data model for the final reporting layer.

In addition, we needed a way to orchestrate the pipelines, a process to manage data lineage and governance, CICD, and monitoring of the entire stack.

In this case, the customer had a requirement to stream the vast amount of their data directly into BigQuery. Rather than pull the data out of BigQuery and do transformations within Dataproc, we decided to use as much SQL logic as possible and only used Dataproc where absolutely necessary.

JOB ORCHESTRATION

Cloud Composer is a managed workflow orchestration tool that allows users to create, schedule, and monitor pipelines within GCP and on-premise. It is built on top of Apache Airflow and uses the Python language for job definitions.

DATA LINEAGE

The Data Fusion offering from Google is used to track the lineage of data as it flows through the pipelines. Metadata can be used to tag different components so that they are easily identifiable and managed. This also helps in discovering new components. For example, you can tag a dataset as experimental or an application as production. These entities can then be discovered by using search queries with the annotated metadata.

Using search, you can discover entities:

- That have a particular value for any key in their properties
- That have a particular key with a particular value in their properties
- That have a particular tag

You can find a dataset or a stream that has a “field with the given name” or a “field with the given name and the given type”.

Lineage can be retrieved for dataset and stream entities. A lineage shows—for a specified time range—all data access of the entity, and details of where that access originated from.

For example, writing to a stream can take place from a worker, which may have obtained the data from a combination of a dataset and a (different) stream. The data in those entities can come from (possibly) other entities. The number of levels of the lineage that are calculated is set when a request is made to view the lineage of a particular entity. In the case of streams, the lineage includes whether the access was reading or writing to the stream.

In the case of datasets, lineage can indicate if dataset access was for reading, writing, or both if the methods in the dataset have appropriate annotations. If annotations are absent, lineage can only indicate that dataset access took place, and does not provide an indication if that access was for reading or writing.

METADATA MANAGEMENT

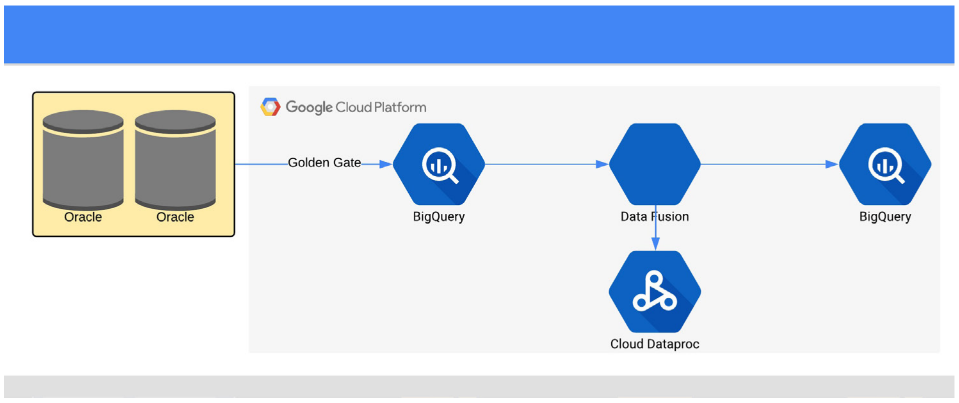
Metadata Management within GCP is done via the Cloud Data Catalog service (Currently in Alpha/Beta). Data Catalog is a centralized and unified data catalog service for all cloud resources, where users and systems can discover data, explore and curate its semantics, understand how to act on it, and help govern its usage.

Data Catalog is a fully managed metadata management service that simplifies data discovery. It offers a search interface, with built-in access level controls. In addition, there is a tool to organize business metadata as schematized tags, enabling discovery of data within their organizations, and fostering a culture of data-driven decision-making.

Data profiling and fingerprinting are performed using the Data Catalog service. The service automatically analyzes data in both BigQuery and Pub/Sub to apply tags and discover similar data throughout the project. In addition, manual tags (via the console and API) can be applied by Ericsson personnel on datasets, data streams, file sets, and other objects.

The service automatically ingests technical metadata for BigQuery and Cloud Pub/Sub and captures business metadata in schematized format via tags, custom APIs, and the UI, offering a simple and efficient way to catalog their data assets.

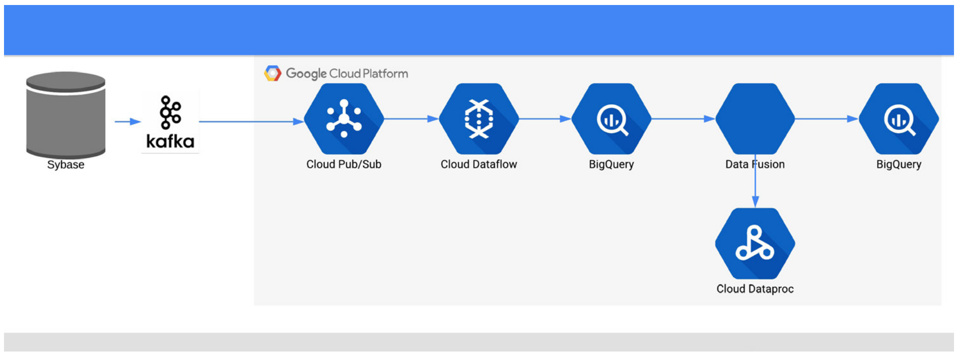
DATA SOURCE #1



Initially, the data will be sent as CDC via the Oracle GoldenGate plugin directly into a BigQuery dataset. The CDC messages will be read by the Data Fusion workflow and placed into a pipeline.

The pipeline will transform the data into the required format. Pythian used a BigQuery Action plugin within Data Fusion to perform this transformation in one step.

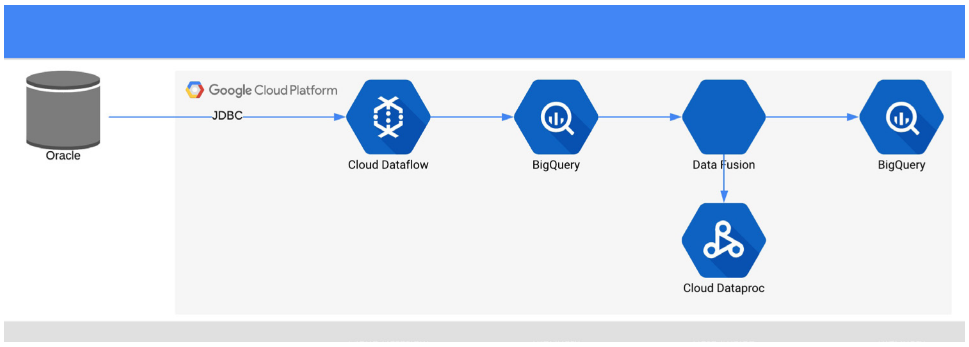
DATA SOURCE #2



Initially, the data will be sent as XML via Kafka and Pub/Sub into a BigQuery dataset using Dataflow as a simple pass-through mechanism. The messages are read by the Data Fusion workflow and placed into a pipeline.

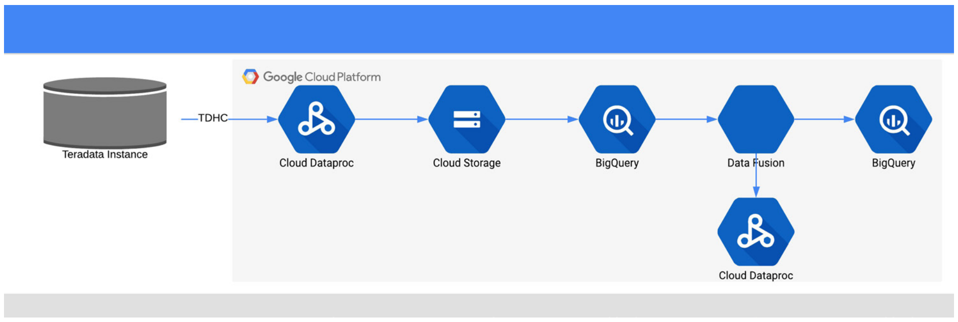
The pipeline transforms the data into the required format. Pythian used a BigQuery Action plugin within Data Fusion to perform this transformation in one step.

DATA SOURCE #3



This is an Oracle source which is imported via a daily batch job. The data is retrieved via Dataflow over a JDBC connection and placed into a BigQuery dataset. Pythian used a BigQuery Action plugin within Data Fusion to perform this transformation in one step.

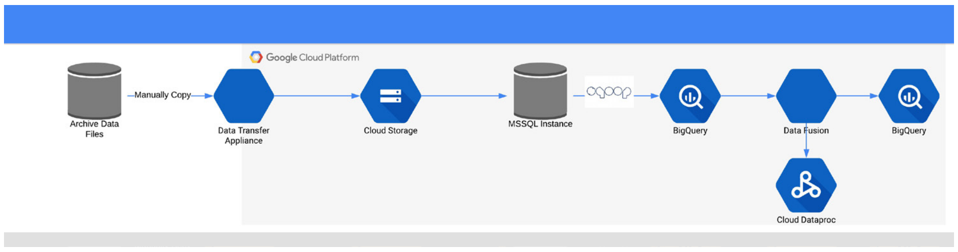
DATA SOURCE #4



The historic Teradata instance needed to be migrated as well. Pythian chose to use the Teradata Hadoop Connector to migrate the data from Teradata into BigQuery. We set up a Dataproc instance with the Hadoop connector installed and wrote scripts to pull the data from the Teradata instance. The data will be pulled over and stored in GCS Buckets, which will then be imported to BigQuery.

We chose to not use the Teradata Data Transfer Service because at the time it did not allow us to write custom queries to pull the data, but required a datetime column to be defined. Several tables within the Teradata instance did not have a datetime column we could use, and so the entire table would have been pulled every time.

DATA SOURCE #5



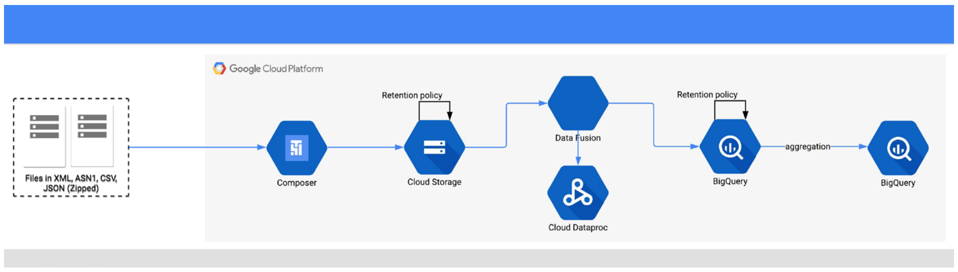
This data source was archived data from Teradata that was not being changed in any way.

We performed the following steps to migrate this data:

- The customer copied the data files onto the Google Transfer Appliance.
- The data was shipped to Google and copied to a GCS Bucket
- Pythian created a SQL Server instance in GCP using the data files.
- Pythian wrote sqoop jobs to export the data and load into BigQuery

After the data was loaded into BigQuery, we were able to reuse the pipelines from data source #2 to move the data into the reporting layer.

DATA SOURCE #6



Finally, the customer had a large set of flat files in various formats which required parsing, transformation, and aggregation.

This data was very large and was not landing directly within BigQuery, so we used a more traditional approach with Dataproc as the main processing engine.

1. The data is pulled from the customer environments using Cloud Composer and SFTP, and the files are stored in Cloud Storage.
2. Pipelines written Data Fusion call Dataproc jobs to read the files and parse them.
   a. As the raw files age, they are moved to long-term, cheaper storage based on policies defined by Ericsson.
3. This job also transforms the data if needed, calculates KPIs, and loads the data into BigQuery.
   a. Similar to GCS Storage, as the data ages, it is placed in cheaper storage automatically.
4. After loading into BigQuery, the data is aggregated and placed into a second set of tables for reporting and KPI measurement.

CONCLUSION

There are plenty of motivations for migrating a Teradata instance to BigQuery. BigQuery has a rapidly growing list of vendors and customers to provide native connections and tooling. The launch of Standard SQL (which is soon to be the default) will make data consumer transitions easier as well. Any data migration process needs to consider the environment, the consumers of the data and the timing.

Data type mapping is one of the more challenging issues, so it's critical to review and plan how data will be transformed as it is migrated into the BigQuery environment.

A migration from Teradata to BigQuery will require a thorough plan and parallelizing Teradata and BigQuery to ensure data quality and consistency, validating the data using reporting and data QA tooling.

As can be seen from this document, a very thorough understanding of the current Teradata environment as well as Google Cloud Platform tools is required for a successful migration. A company planning a lift & shift of their Teradata instance and sources will not see the majority of the benefits in a modern cloud environment.

Pythian recommends a deep dive into the following areas (and more!) to start a project:

- Business Process Logic
- Teradata Data Model(s)
- BTEQ, Stored procedures, and other code
- Pipeline dependencies and scheduling requirements
- Monitoring, auditing, and logging requirements
- Security requirements for the entire stack
- Reporting requirements including KPIs and SLAs
- CICD and development processes

A realistic timeline for most migrations is almost certainly greater than nine months and may be as much as 18 to 24 months.

Pythian's team of experts have Teradata and Google Cloud expertise, as well as decades of combined experience. We leverage our expertise to help you maximize your throughput and mitigate risks while providing the best, custom solution for you and your business.

Contact us to find out how Pythian's analytics experts can help you meet your goals faster, by aligning your business needs and your data strategy and technology. Ask about special pricing on our Teradata to Google Cloud Migration program offerings until August, 2019.

THE AUTHORS

Ekaba Bisong
Data Developer/ Scientist, Google Cloud Data Engineer

Gurinderbeer Singh
Data Developer

John Schulz
Principal Consultant

Kartick Sekar
Solutions Architect

Paul Melanson
Big Data Team Manager

Scott McCormick
Google Cloud Certified Solution Architect

Vladimir Stoyak
Principal Consultant for Big Data, Google Cloud Platform Qualified Solution Developer