Migration of AWS Redshift to Google BigQuery with BigLake

Suhas Hanumanthaiah

suhas.h@hotmail.com

Abstract:
As organizations adopt multi-cloud strategies to optimize performance, cost, and flexibility, seamless migration between cloud data warehouse platforms becomes critical. This research explores the comprehensive process of migrating from Amazon Redshift to Google BigQuery, emphasizing the use of BigLake to enable secure, efficient, and scalable data access without relying on traditional Extract, Transform, Load (ETL) processes. The study outlines the architectural principles, phased migration strategy, implementation methodology, and automation techniques necessary for a successful transition. It highlights the use of BigLake external tables and secure BigQuery Omni VPN connections to directly access and query data stored in Amazon S3, significantly reducing network egress costs and development overhead. Practical considerations for handling complex data types, regional constraints, and schema conversions are addressed, along with a comparison of traditional migration approaches using ETL tools versus the proposed BigLake-based model. The paper demonstrates that using BigLake can cut developer effort by several weeks, reduce migration costs by minimizing data movement, and maintain compliance through secure, trusted pathways. The findings advocate for BigLake as a strategic solution for enterprises aiming to modernize their data architecture and leverage cloud-native analytics in a hybrid or multi-cloud environment.

Keywords: Cloud Data Migration, Amazon Redshift, Google BigQuery, BigLake, Multi-Cloud Strategy, Data Warehouse Modernization

1. INTRODUCTION

Google BigQuery emerges as a pivotal solution for managing and analyzing extensive datasets, addressing the limitations of traditional database management systems in handling large data volumes, which offers SQL-like querying capabilities against massive datasets while operating in the cloud [1]. This cloud-based platform provides real-time insights, enabling users to efficiently analyze and derive value from their data [1]. The evolution of data warehousing solutions like Amazon Redshift has significantly influenced the analytics domain [2]. However, the increasing demand for scalable, cost-effective, and fully managed solutions has led organizations to explore alternatives such as Google BigQuery [3].

With many organizations going through Digital Transformation, rethinking traditional practices and redesigning and automating processes with latest technologies is key to ensuring business efficiency and growth [4]. Organizations are implementing multi-cloud strategy to use best services from different cloud provider. Google BigQuery is prominent and popular data warehouse based on Columnar storage and Massively Parallel Processing. It offers advantages such as scalability, cost efficiency, and seamless integration with other Google Cloud services. Migrating from Amazon Redshift to Google BigQuery can enable organizations to leverage these benefits, improving their data analytics capabilities and reducing operational overhead [6].

This paper is going to explore the strategic considerations and practical steps involved in migrating from Amazon Redshift to Google BigQuery, focusing on leveraging BigLake for enhanced data accessibility and unified governance across different storage formats and locations.

2. LITERATURE REVIEW

1. Although both Amazon Redshift and Google BigQuery are prominent data warehousing solutions built on columnar storage and Massive Parallel Processing (MPP) architecture designed for efficient processing and analysis of large datasets, table design and performance management is different [1].

2. Google recommends using BigQuery data transfer service to load data into BigQuery in a scheduled and managed manner [7]. Redshift is connected through JDBC connection and data transfer internally uses S3 Bucket for extract and load operation. Drawback with this approach is data transfer service uses CSV file format to move data and does not capture all metadata, data types and table relationships. For very large tables, Google recommends transferring one table at a time since BigQuery has a load quota of 15 TB for each load job.[8]

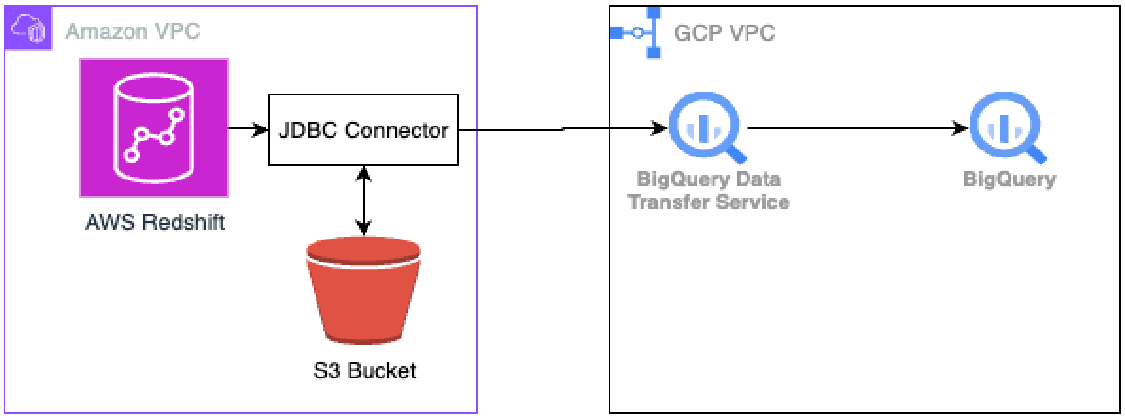

Fig 1: Migration of Redshift to BigQuery using BigQuery Data Transfer Service

3. Google also recommends for migrating data from AWS Redshift to BigQuery using ETL Tools with following steps [9]:

1. Export Amazon Redshift data to Amazon S3.
2. Copy data from Amazon S3 to Cloud Storage by using any of the following options using Storage Transfer Service
3. Transform and then load your data into BigQuery by using any of the following ETL/ELT options:

○ Dataproc
- Read from Cloud Storage with Apache Spark
- Write to BigQuery with Apache Spark
- Hadoop Cloud Storage Connector
- Hadoop BigQuery Connector

○ Dataflow
- Read from Cloud Storage
- Write to BigQuery
- Google-provided template: Cloud Storage Text to BigQuery

○ Cloud Data Fusion
- Read from Cloud Storage
- Write to BigQuery

○ Dataprep by Trifacta
- Read from Cloud Storage
- Write to BigQuery

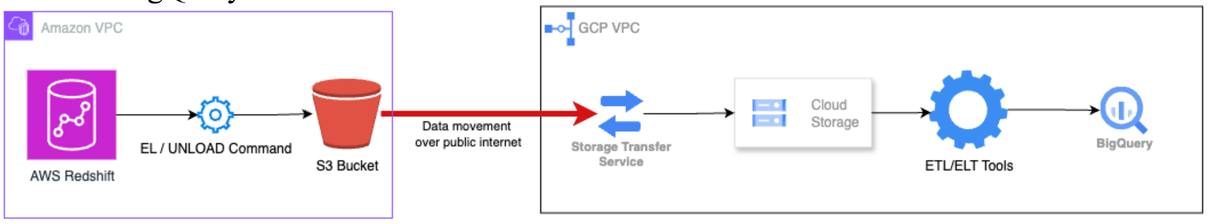

Fig 2: Migration of Redshift to BigQuery using ETL Tools

Drawback with this approach is that the ETL/ELT needs to be developed using third party tool or custom code. Also, the data between Amazon S3 and Google Cloud Storage needs to be transferred over public internet.

3. METHODOLOGY AND APPROACH

The migration from Amazon Redshift to Google BigQuery necessitates a meticulously planned and executed methodology, ensuring a seamless transition and minimal operational disruption, which comprises several key phases, each designed to address specific aspects of the migration process.
BigLake offers a unified data access layer, enabling BigQuery to directly query data stored in various formats and locations, including object storage systems like Amazon S3 and Google Cloud Storage. BigLake is the next evolution of BigQuery Omni, with capabilities such as workload management, enhanced performance, and support for open-source formats using the Apache Iceberg integration. The architecture involves configuring BigLake to access data stored in Amazon S3, allowing BigQuery to directly query the data without the need for extensive ETL processes.

3.1. Architecture

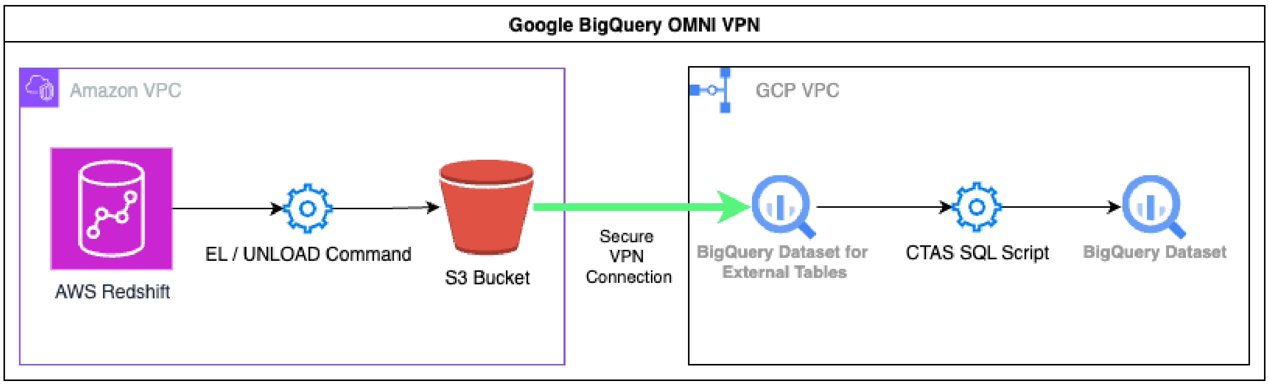

Fig 3: Migration using BigLake

The methodology for migrating from Amazon Redshift to Google BigQuery using BigLake involves the following key steps:

1. UNLOADING AWS Redshift table to AWS S3 Bucket.

a. Create new AWS S3 Bucket with Encryption enabled
b. Use following sample UNLOAD command to extract all necessary table data to S3 Bucket:

```
UNLOAD ( 'SELECT id,name, ST_ASText(region) region, details FROM public.geospatial_data')
to 's3://my-bucket-name-20230101/with_encrypt/public/geospatial_data/'
iam_role 'arn:aws:iam::123456789123:role/service-role/RSCommandsAccessRole'
PARQUET
ALLOWOVERWRITE;
```

Here we are unloading table data from public.geospatial_data to S3 Bucket using IAM Role with PARQUET file format. ALLOWOVERWRITE option allows overwriting the files in S3 Bucket on subsequent execution. Important considerations, for GEOMETRY and GEOGRAPHY data types, use ST_ASText function to turn the values into string before migration.

2. Create External Connection in Google Cloud Platform to connect to Amazon S3 Bucket.

3. Create BigLake external table on top of the AWS S3 Bucket using the external connection from previous step. It is recommended to place them in a separate Dataset to distinguish permissions.

```
CREATE EXTERNAL TABLE redshift.geospatial_data
 WITH CONNECTION aws-us-east-1.aws_s3_bucket
 OPTIONS (
  FORMAT="PARQUET",
  URIS=["s3:// my-bucket-name-20230101/with_encrypt/public/geospatial_data/*.parquet"],
  MAX_STALENESS = INTERVAL 1 DAY,
METADATA_CACHE_MODE = 'AUTOMATIC' );
```

4. BigLake uses secure BigQuery Omni VPN connection. Omni uses a QUIC-based [5] zero-trust VPN. This VPN enables network endpoints hosted outside of Google production data centers to transparently communicate with services within Google.
5. Use CREATE TABLE AS SELECT statement to copy the table from external BigLake table into BigQuery managed storage.

```
CREATE OR REPLACE TABLE `bigquery-project-name`.`public`.`geospatial_data` AS select * from redshift.geospatial_data ;
```

3.2. Implementation Process and Steps

Migrating large data warehouse can be challenging because of the volume of data. We need to come up with strategy to migrate tables in phased approach. Tables can be categorized based on update frequency like, Static Tables, Monthly Updates, Weekly Updates, Daily Updates, and real-time updates.

Phase 1: Migrate Static Tables and Monthly Updated Tables using Full Load Strategy

- Since these tables change less frequently, we can move out all data during off-peak hours, minimizing disruption.
- On final Production cut-over, we need to validate the row counts between source Redshift table and destination BigQuery table, if not matched then re-extract them.

Phase 2: Migrate Weekly Updated Tables, Daily Updated Tables and Real-time Updated Tables using Delta Load Strategy

- Modify the Source query to extract data based on date or auto number integer field so that only the delta records will be extracted.
- By ensuring the UNLOAD Query is parametrized with no overlaps, duplicate records will be avoided and we can implement incremental insert of data on final production cut over

To automate the generation of BigLake external tables in Google Cloud Platform from existing tables in Amazon Redshift, information from system tables such as `INFORMATION_SCHEMA.TABLES` can be leveraged, which allows the extraction of table schemas, and descriptions, which are crucial for accurately recreating the table structures in BigQuery.

```
with vars as (
select cast('s3://testing-suhas-redshift-20250712/with_encrypt/'as varchar(100)) bucket
,cast('arn:aws:iam::506949280622:role/service-role/AmazonRedshift'as varchar(100)) iamrole
)
select
'unload ( "SELECT * FROM ' + t.table_schema + '.' + t.table_name+"')' + chr(10)
+ 'to '"+ v.bucket + t.table_schema + '/' + t.table_name + "'" + chr(10)
+ 'iam_role '"+ v.iamrole + "'" + chr(10)
+ 'PARQUET' + chr(10)
+ 'ALLOWOVERWRITE;' UNLOAD_QUERY
from information_schema.tables t
full join vars v on 1=1
where table_type = 'BASE TABLE'
and table_schema = 'public';
```

The generated query provides the necessary SQL syntax to unload data from Redshift to S3 in Parquet format, which is optimized for analytical queries. It also enables the OVERWRITE permission to overwrite the files on the S3 bucket. The generated query can be grouped into Stored Procedure based on how frequently they are refreshed.

The implementation phase also involves setting up secure connections between Google Cloud and AWS, which is facilitated by BigQuery Omni's QUIC-based zero-trust VPN, ensuring that data transfer is secure and compliant with organizational security policies. This setup allows network endpoints hosted outside of Google's production data centers to communicate transparently with services within Google [10].

3.3. Datatype considerations

In Amazon Redshift UNLOAD command following considerations needs to be followed for GEOMETRY and GEOGRAPHY datatypes:

- Use `ST_AsText` function to turn the values into string before migration.
- On BigQuery side, use `ST_GeogFromText` or `ST_GeomFromText` to turn the string back into Geography or Geometry type.

For SUPER data type:

- No transformation is required before migration. SUPER datatype fields are converted into Byte field in PARQUET file.
- On BigQuery side, the column will be automatically converted to BYTES Datatype [2]. When querying the data, cast the BYTES data to STRING and then to JSON data.

When schema is auto detected from PARQUET file, the width of the STRING columns or Precision and Scale of NUMERIC columns are not detected. It creates a base type without length restriction. This can be resolved by doing below steps.
For Google Managed BigQuery tables, create empty tables with right schema definition before migrating the data.

3.4. Location Considerations

Google BigLake supports only few AWS regions to integrate with. Usually UNLOAD command only works for transferring data from Redshift to S3 in same region. Also, BigQuery Dataset's region is defaulted to Project default Region. But in BigQuery, you can create a dataset pointing to region that is different from default Project Region. Also, Create Table as Select will work to transfer data from different Region. So, if Redshift and BigQuery are hosted in different regions then following changes needs to be considered to the original architecture:

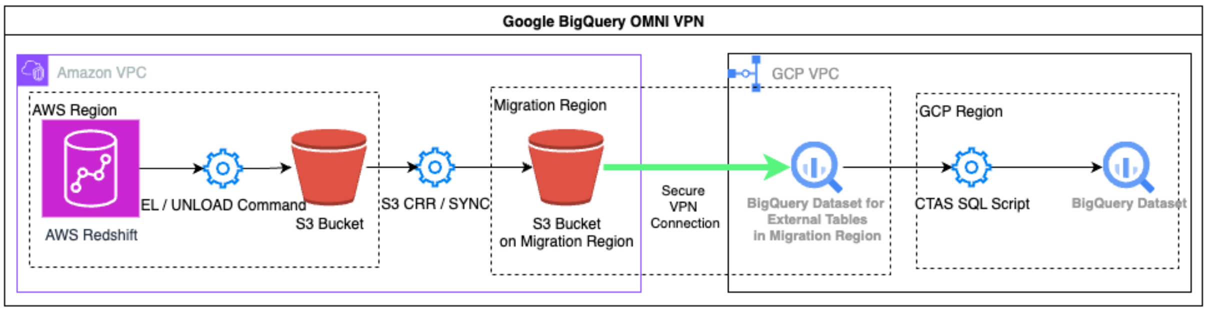

Fig 4: Migration using BigLake in different Region

1. Determine the Migration Region by evaluating regions supported by BigLake and choose the one closest to AWS Region. Let's call this region Migration Region
2. Create BigQuery Dataset in Migration Region. This dataset will work as the landing zone for creation of external table pointing to S3 Bucket
3. Create Additional S3 Bucket in Migration Region dedicated for migration
4. Setup S3 Cross-Region Replication (CRR) or use aws s3 sync command line to move data between S3 buckets.

4. RESULTS

Migrating from AWS Redshift to Google BigQuery using BigLake as compared to using a traditional approach offers several advantages.

1. Cost Efficiency:

Migration of AWS Redshift to Google BigQuery cost is driven by compute, services and storage costs at both Redshift and BigQuery side. Using BigLake provides a great advantage by reducing cost by eliminating the requirement for a provisioned ETL/ELT tool. Also, in this paper we have provided automation code that can migrate all tables from Redshift to BigQuery with zero ETL. These UNLOAD query can be scheduled or turned into Stored Procedures in Redshift.

Based on estimate of migrating 600 tables of total size 100 TB in compressed PARQUET format, the data egress cost from Redshift to Amazon S3 would be approximately $900. By leveraging BigLake, the network egress costs associated with transferring data from Amazon S3 into BigQuery are effectively eliminated, as BigLake facilitates direct querying and processing of data within S3 without necessitating data movement. By eliminating development, deployment and testing of ETL/ELT tool, we are saving at least 3 weeks of developer effort.

2. Security Compliance:
Ensuring that all data governance and compliance requirements are met is critical for successful migration. One of the major drawbacks of other migration process is that the data from S3 to Google Cloud Storage gets copied over internet although both cloud providers are on trusted VPC peering. Using BigLake with BigQuery Omni services on VPN ensures that the data is within trusted VPN network.

5. CONCLUSION

The migration from Amazon Redshift to Google BigQuery using BigLake offers a transformative approach for organizations seeking scalable, cost-effective, and secure data warehousing solutions in a multi-cloud environment. By leveraging BigLake's ability to directly query data stored in external object storage such as Amazon S3, organizations can eliminate traditional ETL bottlenecks, minimize data movement, and streamline the transition process. This paper has outlined a practical and phased methodology that addresses both strategic and technical considerations—including architecture setup, implementation steps, automation of migration queries, and handling of complex data types and region-specific challenges. The proposed approach not only reduces the operational overhead associated with traditional migration methods but also ensures better compliance and security by using VPN-based trusted network pathways through BigQuery Omni. Furthermore, significant cost savings are achieved by avoiding redundant data storage and eliminating the need for custom ETL tool development. By enabling unified data governance and providing automation capabilities, BigLake establishes a modern, efficient framework for data migration and analytics across hybrid cloud environments. This research underscores the viability and advantages of adopting BigLake as a central component in modern data platform architectures, paving the way for seamless, scalable, and secure analytics in the cloud era.

REFERENCES:

[1] S. M. Fernandes and J. Bernardino, "What is BigQuery?," Jan. 2014, doi: 10.1145/2790755.2790797.
[2] N. Armenatzoglou et al., "Amazon Redshift Re-invented," in Proceedings of the 2022 International Conference on Management of Data, Jun. 2022, p. 2205. doi: 10.1145/3514221.3526045.
[3] Md. H. Ali and Md. A. Hossain, "Big Data Analysis using BigQuery on Cloud Computing Platform," Australian Journal of Engineering and Innovative Technology, p. 1, Jan. 2021, doi: 10.34104/ajeit.021.0109.
[4] A. Scheer, F. Abolhassan, W. Jost, and M. Kirchmer, Business Process Automation. Springer Nature, 2004. doi: 10.1007/978-3-540-24702-9.
[5] Adam Langley et al. 2017. The QUIC Transport Protocol: Design and InternetScale Deployment. In SIGCOMM. ACM, 183–196.
[6] S. Ponnusamy, "Evolution of Enterprise Data Warehouse: Past Trends and Future Prospects," International Journal of Computer Trends and Technology, vol. 71, no. 9, p. 1, Sep. 2023, doi: 10.14445/22312803/ijctt-v71i9p101.
[7] B. Abu-Salih, P. Wongthongtham, D. Zhu, K. Y. Chan, and A. Rudra, "Introduction to Big Data Technology," in Springer eBooks, Springer Nature, 2021, p. 15. doi: 10.1007/978-981-33-6652-7_2.
[8] "Migrate schema and data from Amazon Redshift." Accessed: Jul. 01, 2023. [Online]. Available: https://cloud.google.com/bigquery/docs/migration/redshift#set-up-transfer
[9] "Amazon Redshift to BigQuery migration: Overview." [Online]. Available: https://cloud.google.com/bigquery/docs/migration/redshift-overview#migration_using_pipelines
[10] S. Deochake, V. Channapattan, and G. B. Steelman, "BigBird: Big Data Storage and Analytics at Scale in Hybrid Cloud," arXiv (Cornell University), Jan. 2022, doi: 10.48550/arxiv.2203.11472.

7. Abbreviations

Abbreviation	Full Form
AWS	Amazon Web Services
CRR	Cross-Region Replication
CTAS	Create Table As Select is query to create a table from select query
EL	Extract, Load
ELT	Extract, Load, Transform
ETL	Extract, Transform, Load
GCP	Google Cloud Platform
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
MPP	Massively Parallel Processing
S3	Simple Storage Service (AWS)
SQL	Structured Query Language
VPC	Virtual Private Cloud
VPN	Virtual Private Network