**SNOWFLAKE**

**Snowflake** is a**Fully managed Warehousing Service** that enables clients to connect, load, analyze, and securely share their data while allowing for near-infinite concurrent operations scalability. Some of the most common use cases include Data Lakes, Data Engineering, Data Application Development, Data Science, and safe Data Consumption.

**SNOWFLAKE ARCHITECTURE**

Snowflake is a cloud data warehouse built on top of public cloud infrastructure like AWS, AZURE and is a true SaaS offering. There is no hardware (virtual or physical) for you to select, install, configure, or manage. There is no software for you to install, configure, or manage. All ongoing maintenance, management, and tuning is handled by Snowflake. Architecturally there are three main components that make up the Snowflake data warehouse. The three main components are:

1. **Storage layer** is where the snowflake data is stored which comprises of the external provider such AWS, GCP and AZURE depending on which platform it's hosted on.
2. **Query processing layer** is the **muscle of the snowflake**, it’s where the query processing takes place within the virtual warehouse.
3. **Cloud service layer** also known as the **Brian of the snowflake**. This manages Security, authentication and authorization, metadata management, infrastructure management and Optimization.

A blue and white diagram with words

Description automatically generated with medium confidence

**SNOWFLAKE FEATURES.**

**1 ZERO COPY CLONE**

Zero-Copy Cloning (sometimes called simply "cloning") is a Snowflake feature that makes a copy of a database/ database Objects, without duplicating the data it contains. The clone operation takes a snapshot of the source data when the clone is created and makes this data available to the cloned object. After this point, the clone is independent of the source, so any subsequent changes made to either the source or the clone aren't reflected in the other.

* However, External and Internal (Snowflake) stages object types are *not* cloned:

*Create or replace table <clone\_table\_name> CLONE <source\_table\_name>;*

**USE CASE:**

1. Clone PROD to DEV just like a REFRESH
2. Clone a database for a Backup. It's doesn't occupy extra space.
3. We also use CLONE for Time Travel

**2 Time Travel and Fail Safe**

Snowflake Time Travel enables accessing historical data (i.e. data that has been changed or deleted) at any point within a defined period. It serves as a powerful tool for performing the following tasks:

* Query data in the past that has since been updated or deleted.
* Create clones of entire tables, schemas, and databases at or before specific points in the past.
* Restore tables, schemas, and databases that have been dropped.

Fail-safe on the other hand provides a (non-configurable) 7-day period during which historical data may be recoverable by Snowflake.

* This period starts immediately after the Time Travel retention period ends.
* Long-running Time Travel query will delay moving any data and objects (tables, schemas, and databases) in the account into Fail-safe, until the query completes.

**3 Support for Structured and Semi Structured Data**

* You can load CSV, Parquet, orc, avro, json and xml files.

**4 SNOWPIPE**

Snowflake has a serverless component called SOWPIPE, which can be integrated with external objects storage like S3, azure blob storage and ingest data when new data arrives in the storage

**STEPS**

* Create a stage for the connection (Storage integration Objection)
* Test copy command to make sure it works
* Create pipe as object with COPY COMMAND
* Setup **S3 or Azure Blob Storage Notification to trigger snowpipe**.

**5 Micro Partition and Data Clustering**

* Snowflake implements a powerful and unique form of partitioning called micro-partitioning, that delivers all advantages of static partitioning
* All data in Snowflake tables is automatically divided into micro-partitions which are contiguous units of storage. Each micro partition contains between 50 and 500 MB of uncompresses data
* Tables are transparently partitioned using the ordering of the data as it is inserted/loaded into tables

**6 Security**

* All ingested data stored in Snowflake tables is encrypted using AES-256 strong encryption.
* All files stored in internal stages for data loading and unloading operations is automatically encrypted using AES-256 strong encryption.
* Snowflake supports row-level security using row access policies to determine which rows to return in the query result.

**7 Data swapping**

* It is possible to swap all the content, metadata between two specified tables, including any integrity constraints defined for the tables in one transaction. Also, swap all access control privilege grants.
* Swapping the tables does not update references to those tables in other objects (such as view definitions), so this must be updated to the new names manually

e.g Alter table table\_1 swap with table\_2

**Snowflake vs Postgres:**

**Functionality**

Snowflake is a Cloud-Based Data Warehousing service that works with Semi-Structured and Structured data with a commercial license. PostgreSQL is an Open-Source RDBMS that was developed to serve as a general-purpose entity. Both PostgreSQL and Snowflake are ACID compliant. Snowflake’s ability to separate compute and storage through virtual data warehouses gives it an edge over PostgreSQL.

**Scalability**

PostgreSQL and Snowflake both focus on scalability with Snowflake allowing you to run massively concurrent workloads in a single system at scale (MPP), the only Cloud data warehousing solution to do so. PostgreSQL’s cluster-based solutions allow for immense expansion. The WAL files of PostgreSQL have a 16 MB limit that allows you to foster integrity during the scalability operations.

**Support**

In terms of support, both Snowflake and PostgreSQL have dedicated communities of developers offering videos, articles, tutorials, and various other resources to help you out on your journey.

**Security**

In terms of security, PostgreSQL offers fine-grained access rights following the SQL standard. Snowflake, on the other hand, offers users a fine-grained authorization concept, pluggable authentication, and user roles among an assortment of features which makes Snowflake a more viable choice concerning security features.

Pricing

PostgreSQL is offered free of cost. Snowflake, on the other hand, lets you pay for the compute and storage resources that you leverage. You can start with a 30-day free trial after which you can opt for one of the following pricing plans to suit your needs:

**GENERAL MIGRATION STEPS FROM ON-PREM POSTGRES TO SNOWFLAKE IN AZURE CLOUD**

Migrating an on-premises database warehouse to Snowflake involves several steps, from planning and preparation to data migration and post-migration tasks. Here is a high-level overview of the process:

1. **Assess and plan:**

* Analyze your current on-premises database to understand its size, schema, data types.
* need special handling in Snowflake. Identify any potential challenges, such as data type conversions or custom functions that might
* Choose the appropriate Snowflake edition and features based on your requirements.

1. **Prepare the environment:**

* Set up a Snowflake account and create the necessary virtual warehouses, databases, and schemas.
* Configure any required integrations, such as data lakes or external data sources, with Snowflake.
* Establish connectivity between on-premises environment and Snowflake using tools like SnowSQL or third-party connectors.

1. **Schema migration:**

* Convert on-premises database schema to a Snowflake-compatible schema. This involve adjusting data types, creating new tables, or modifying table structures.
* Use tools like SnowConvert or manual scripting to convert stored procedures, functions, and other database objects to Snowflake-compatible objects.
* Load/Create the converted schema objects into Snowflake.

1. **Data migration:**

* Extract data from your on-premises postgres database using data export tools (pg\_dump) or custom scripts.
* Stage the extracted data in Azure Blob Storage
* Load the staged data into Snowflake using Snowpipe or the COPY INTO command.
* Validate the data migration by comparing row counts, data types, and sample data between the source database and Snowflake.

* **Switch to production:**
* Schedule a cutover window to switch from on-premises database to Snowflake.
* Update any applications, reports, or data pipelines to use the new Snowflake data warehouse.
* Monitor performance and address any issues that arise during the transition.

1. Post-migration tasks:

* Train your team on using and managing the new Snowflake environment.
* Continuously monitor and optimize your Snowflake environment for cost, performance, and security.

**Task:**

**As the lead DBA, my task was to design and implement a Snowflake-based solution that would make several layers of data available to different teams while ensuring security, performance, and scalability. This involved migrating the data from various sources, creating the necessary data models and views, and setting up appropriate access controls.**

**Action:**

1. Collaborated with stakeholders to identify data requirements and establish a clear understanding of the data needed for each team.

1. Evaluated and selected the best Snowflake edition, in this case, the "Enterprise" edition, that would meet our performance and security requirements.

1. Migrated data from various sources, such as relational databases Oracle, NoSQL databases, and CSV files, into Snowflake using tools like SnowSQL and Snowpipe.

1. Designed a multi-layered data architecture that included raw data, cleansed data, and aggregated data layers. This allowed each team to access the appropriate level of detail for their specific use-cases.

1. Created virtual warehouses in Snowflake to isolate compute resources for each team, ensuring optimal performance and cost-efficiency.

1. Implemented data sharing and data governance best practices by using Snowflake's role-based access control (RBAC) to grant the necessary privileges to each team.

1. Set up automated data refresh mechanisms, using tasks and streams, to ensure that the data in Snowflake was always up-to-date and consistent with source systems.

1. Monitored and optimized the Snowflake environment for performance, query optimization, and cost management.

1. Documented the entire implementation process and provided training to the teams on how to use Snowflake effectively.

Result: By leveraging Snowflake, we were able to make several layers of data available for reporting, data science, and analytics, while maintaining security and performance. This solution enabled faster insights and improved collaboration across teams. Moreover, the cloud-based nature of Snowflake allowed us to easily scale our data infrastructure as our company grew, and we saw a significant reduction in overall data management costs.

**Migrating from on-prem PostgreSQL database to Snowflake involves several steps, including extracting data from the PostgreSQL preparing the data for import, and loading the data into Snowflake. Here is a step-by-step guide process:**

1. **Preparation:**

* Ensure that you have a Snowflake account with the appropriate permissions to create and manage databases, tables, and other objects.
* Identify the PostgreSQL tables and schema to be migrate to Snowflake.

1. **Export data from PostgreSQL:**

* Use a tool like pg\_dump to export the data from PostgreSQL.
* --data-only flag to extract only the data, without the schema definitions. For example:   
   pg\_dump --data-only--dbname=your\_database --username=your\_username --host=your\_host --port=your\_port --file=data\_dump.sql

1. **Upload the exported data to a staging area:**

* Create a staging area for the data by uploading the exported data to a cloud storage like an Amazon S3 bucket or Azure Blob
* If necessary, you can compress the exported data files using gzip, as Snowflake supports loading compressed files.

1. **Create the target schema in Snowflake:**

* Use the Snowflake Web Interface or SnowSQL command line to create the target database and schema in Snowflake that will store the migrated data. For example:   
  CREATE DATABASE your\_database;

CREATE SCHEMA your\_schema;

* Recreate the table structure from PostgreSQL instance in Snowflake. Choose the appropriate data types for each column, taking into account Snowflake's supported data types.

1. **Create a Snowflake stage:**

* Create a stage in Snowflake to reference the location of the exported data files in your cloud storage. For example, to create a stage:   
     
  CREATE  STAGE your\_stage

URL='blob storage container url/'

CREDENTIALS = ( AWS\_KEY\_ID = 'your\_blob\_key\_id',

AWS\_SECRET\_KEY = 'your\_aws\_secret\_key'

);

1. **Load data into Snowflake:**

* Use the COPY INTO command to load the data from the stage into the target Snowflake table. For example:   
     
  COPY INTO your\_database.your\_schema.your\_table

FROM @your\_stage

FILE\_FORMAT = (

       TYPE ='CSV',

       FIELD\_DELIMITER =',',

        FIELD\_OPTIONALLY\_ENCLOSED\_BY ='"',

      SKIP\_HEADER =1

)

ON\_ERROR ='CONTINUE';

1. **Validate the migration:**

* Once the data is loaded, run queries on the Snowflake tables to ensure that the data has been migrated correctly. Compare the row counts, data types, and data values between the PostgreSQL instance and the Snowflake tables.