SNOWFLAKE ASSIGNMENT

Name: Polireddi Govind **Employee Id:** AS1552

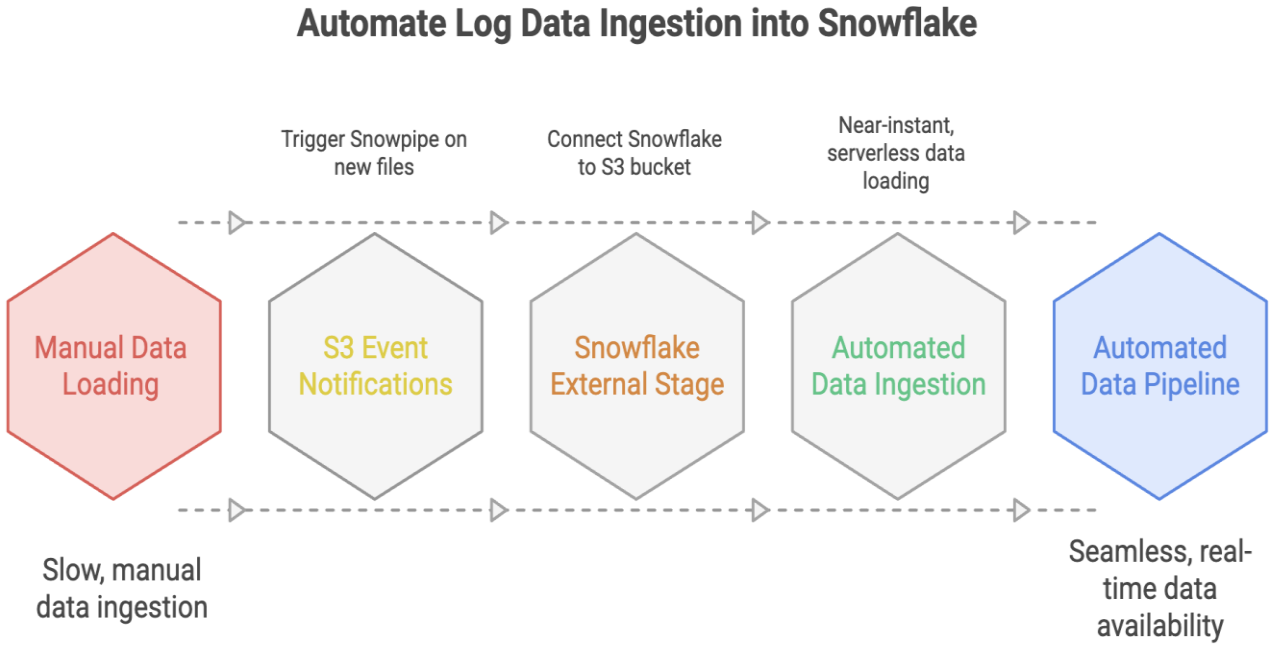
Problem Statement 1:

Automate Real-time Log Data Ingestion from AWS S3 into Snowflake using Snowpipe and Monitor via Snowsight

Topics Covered: Snowpipe, Internal/External Stage, Copy Options, Streaming, SnowSQL, Performance Optimization, Virtual Warehouse, Caching, Monitoring (Snowsight), AWS S3 Integration, Time Travel, Clustering, Data Sampling

Project Overview:

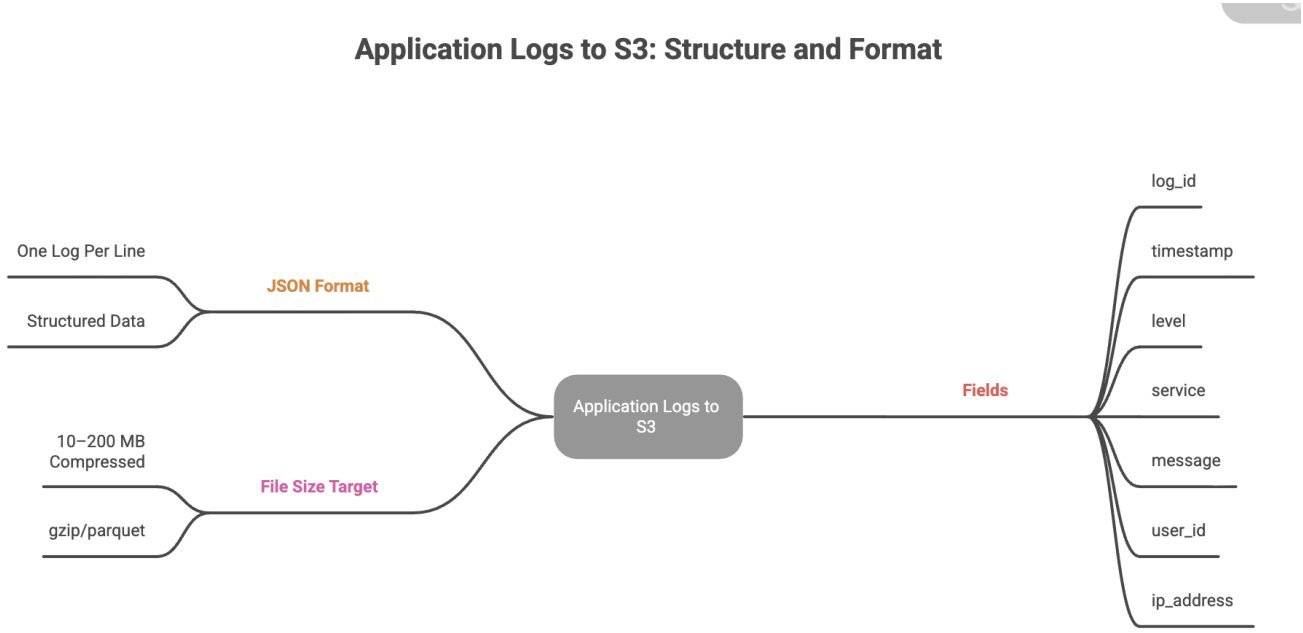
This project aims to automate real-time log data ingestion from AWS S3 into Snowflake using Snowpipe, establishing a seamless, serverless, and near-instant data loading pipeline. By connecting Snowflake with S3 event notifications, newly uploaded log files are automatically ingested, processed, and made available for analytics with minimal manual effort. The solution focuses on performance optimization, cost efficiency, and operational visibility, leveraging Snowflake's capabilities such as external/internal stages, copy options, virtual warehouses, clustering, caching, time travel, and Snowsight monitoring.



Architecture Overview

1. Log Producers → AWS S3

- Application logs (e.g., auth, payments, orders, shipping) are written to S3 in **JSON format**, one log per line.
- Each JSON contains fields like:
    - log_id, timestamp, level, service, message, user_id, ip_address.
- File size target: **10–200 MB compressed (parquet)** for efficient Snowpipe ingestion.



2. S3 Event Notifications → Snowflake Snowpipe

- Each new file triggers an **S3 event notification (via SNS/SQS)**.
- Snowpipe automatically loads data into Snowflake without manual intervention.

3. Snowflake Staging and Curation

- **External Stage:** Secure pointer to the S3 bucket/prefix.
- **Snowpipe:** Automatically ingests JSON logs into a **raw staging table**.
- **Curation Layer:** Transforms, normalizes, and deduplicates data into curated analytics tables.

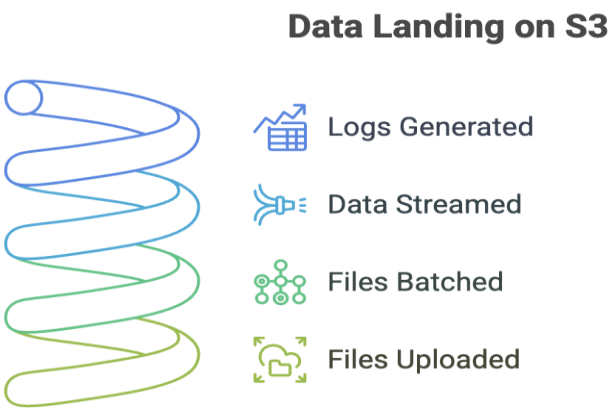
4. Monitoring and Optimization

- **Snowsight:** Monitors ingestion latency, file load history, and error counts.
- **Virtual Warehouses:** Handle downstream transformations.
- **Clustering & Time Travel:** Improve query speed and allow recovery from data issues.

Step-by-Step Approach for Implementation

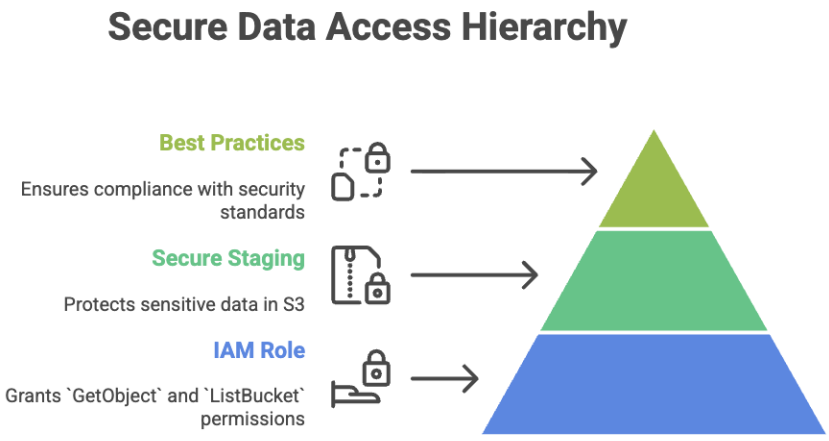
Step 1: Data Landing on S3

- Logs generated by different services are streamed to the S3 bucket.
- Files are batched and upload to the S3 bucket.



Step 2: Setting Up the Snowflake Stage

- Create an **AWS IAM role** with GetObject and ListBucket permissions.
- Stage points securely to the S3 log location.

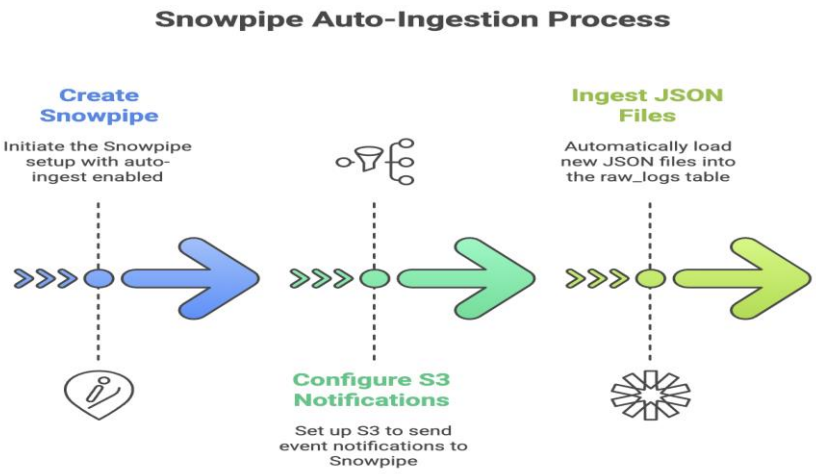


Step 3: Defining File Format and Copy Options

- Define a **JSON file format** to handle the log data structure:
- COPY options:
    - Use ON_ERROR='CONTINUE' to skip corrupted records.
    - Enable PURGE=FALSE for reprocessing flexibility.

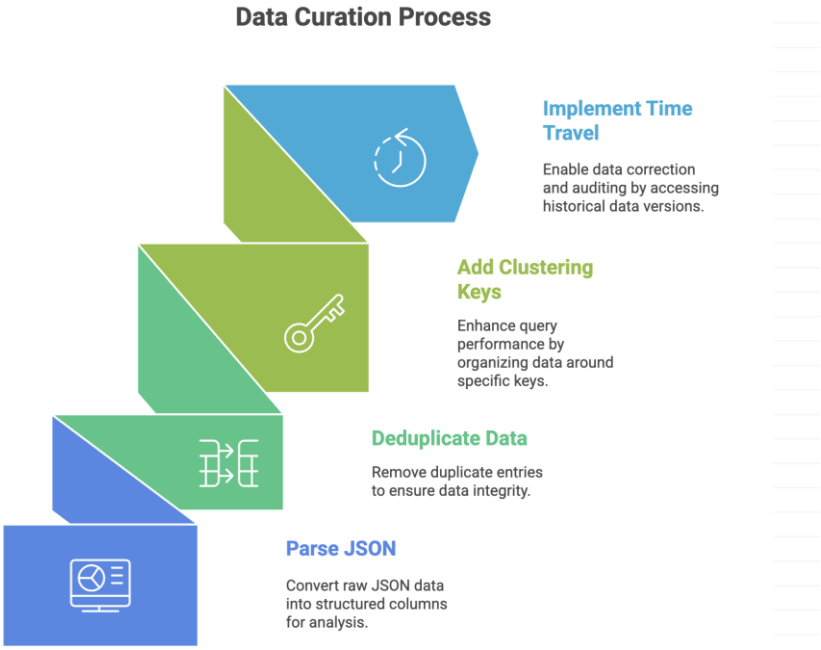
Step 4: Automating Ingestion with Snowpipe

- Create a **Snowpipe** with AUTO_INGEST = TRUE:
- Configure S3 event notifications to publish to the Snowflake pipe endpoint.
- Each new JSON file is ingested into the raw_logs table.



Step 5: Transforming and Curating Data

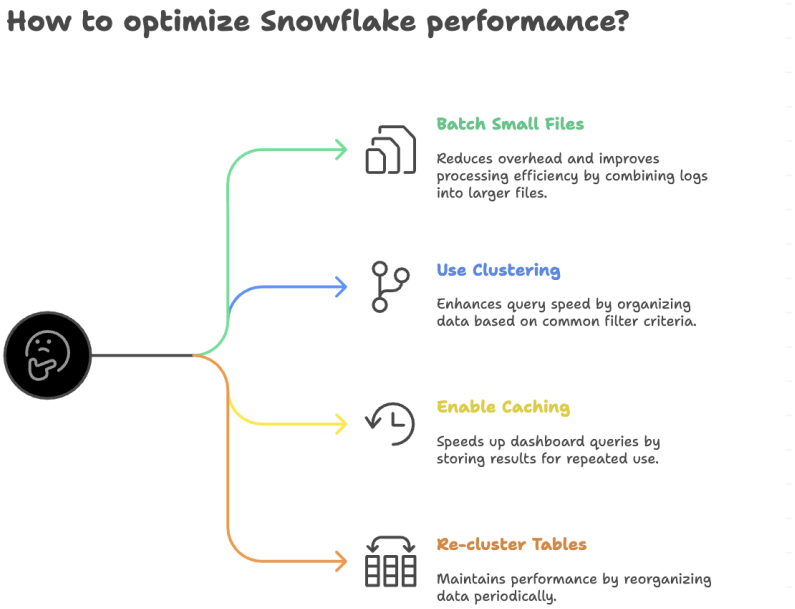
- Parse JSON data into structured columns:
- Apply **deduplication** using DISTINCT or a **merge on** log_id.
- Add **clustering keys**: (event_time, service_name).
- Use **Time Travel** for data correction and auditing.



Step 6: Performance Optimization

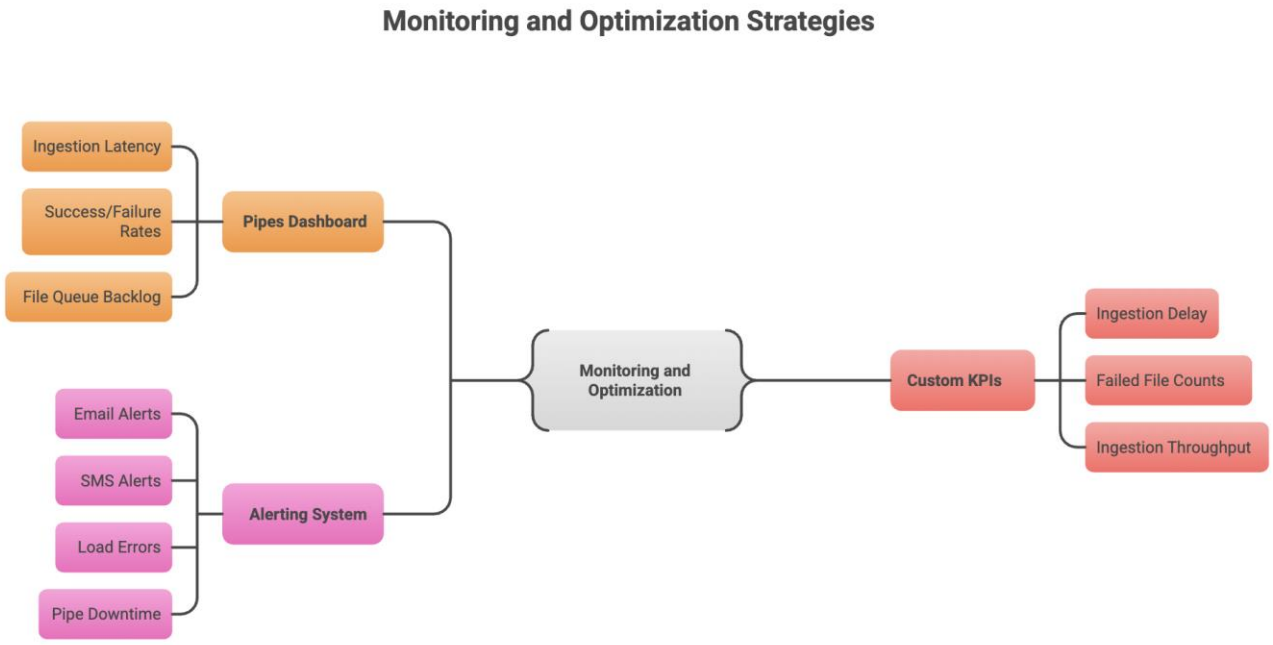
- Avoid small files — batch logs into optimal file sizes.
- Use **clustering** and **search optimization** for fast filtering (e.g., by service_name or log_level).

- Enable **caching** for repeated dashboard queries.
- Periodically **re-cluster** curated tables to maintain performance.



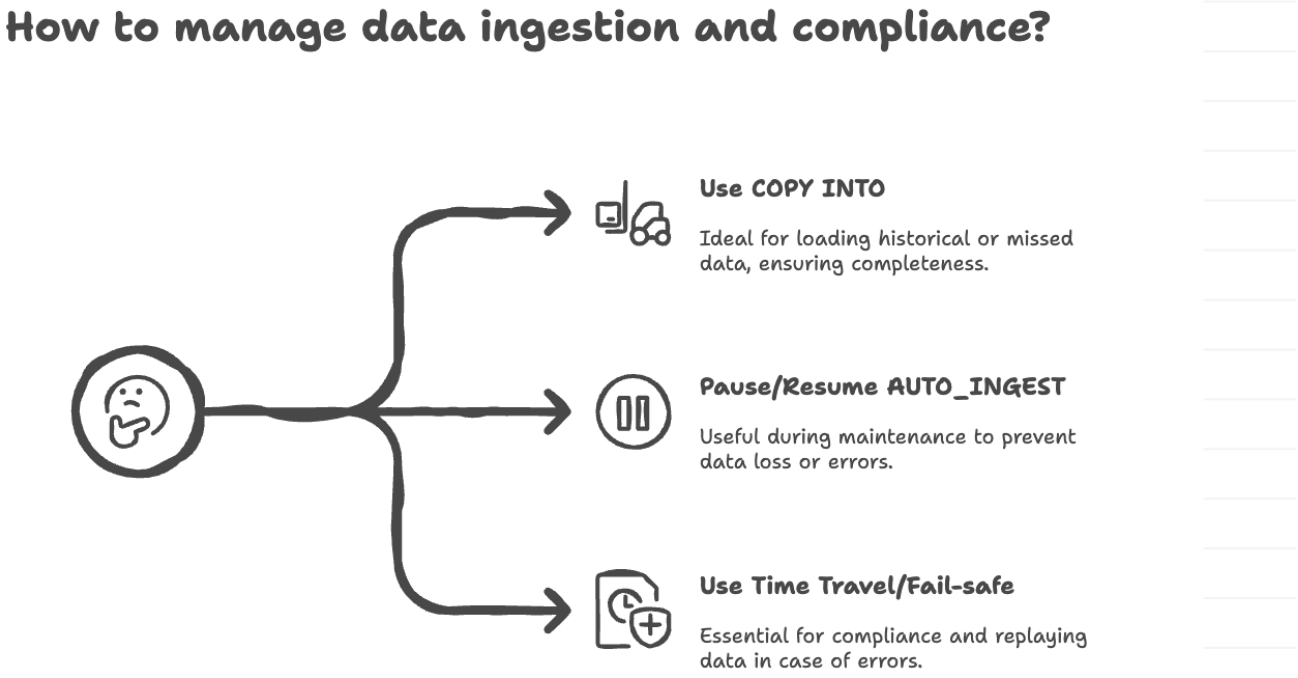
Step 7: Monitoring with Snowsight

- Use the **Pipes Dashboard** to track:
    - Ingestion latency
    - Success/failure rates
    - File queue backlog
- Set up **custom KPIs**:
    - Ingestion delay = CURRENT_TIMESTAMP - file_last_modified
    - Failed file counts, ingestion throughput
- Configure **email/SMS alerts** for load errors or pipe downtime.



Step 8: Backfills and Reprocessing

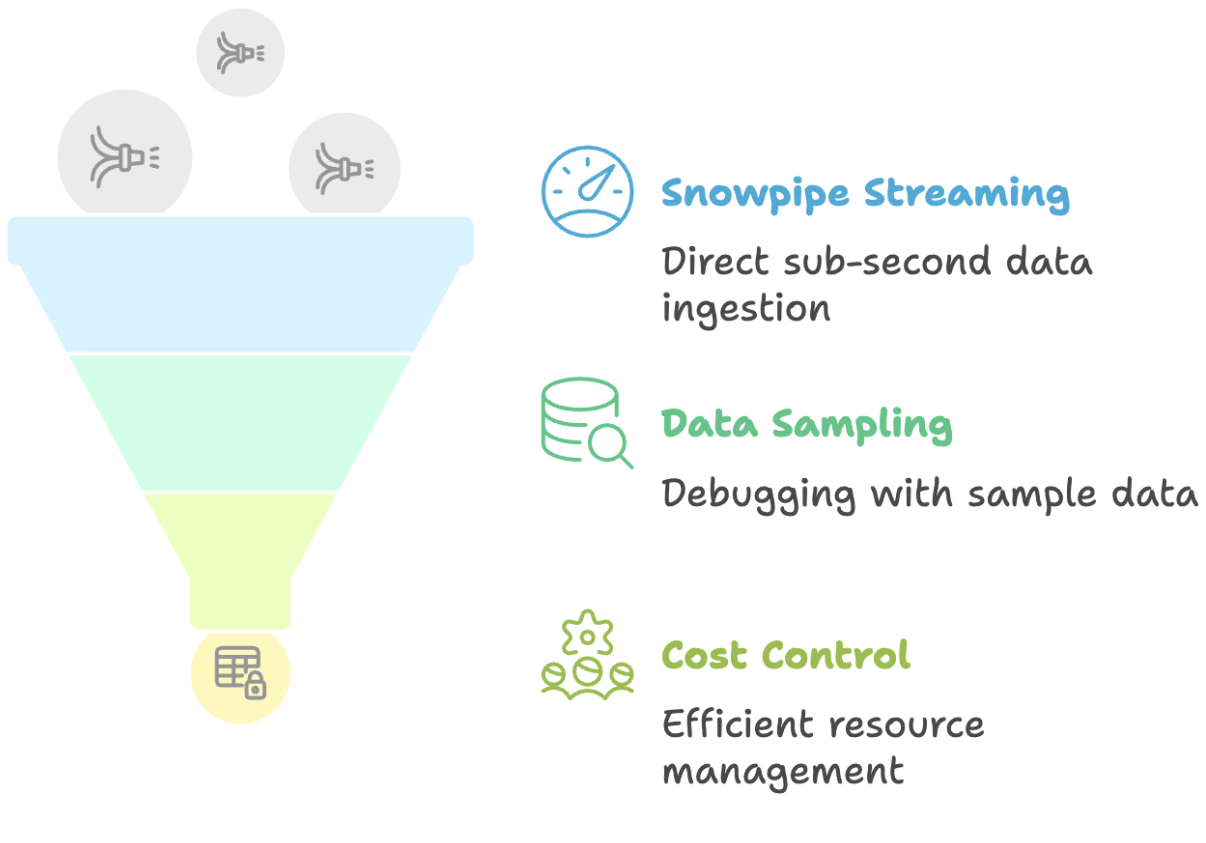
- Use **COPY INTO** for historical or missed data loads:
- Pause/resume AUTO_INGEST during maintenance.
- Use **Time Travel** and **Fail-safe** for replay or compliance scenarios.



Step 9: Advanced Capabilities

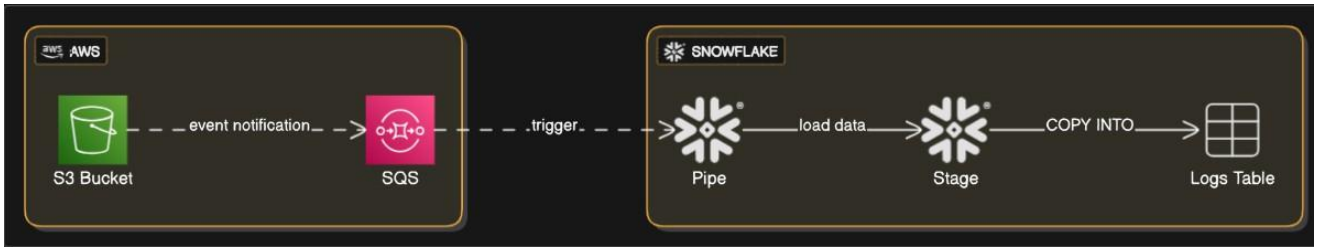
- **Snowpipe Streaming:** For sub-second ingestion directly from producers (optional).
- **Data Sampling:** Use SAMPLE or partition filters for debugging.

- **Cost Control:**
    - Use Snowpipe for ingestion (Snowflake-managed compute).
    - Use auto-suspend warehouses for transformation to minimize compute cost.



ARCHITECTURE:

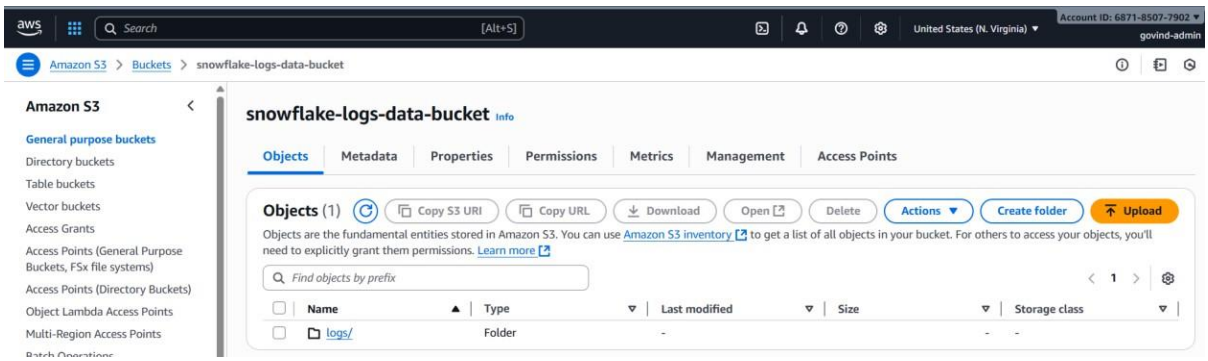
S3 → SQS → Snowpipe → Snowflake Stage → Copy Into Logs Table (Snowflake)



IMPLEMENTATION

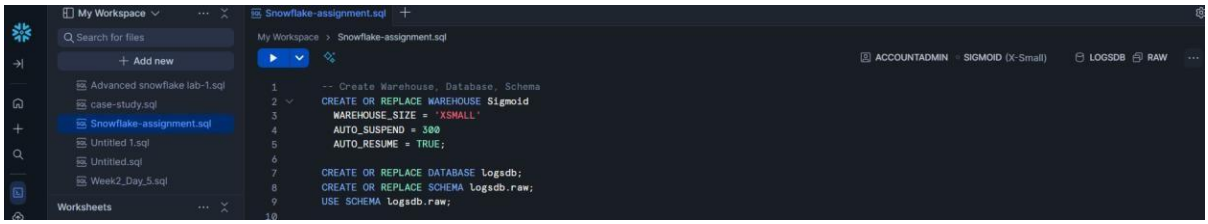
S3 BUCKET:

- ✓ Created a S3 bucket “snowflake-logs-data-bucket”



SNOWFAKE WAREHOUSE:

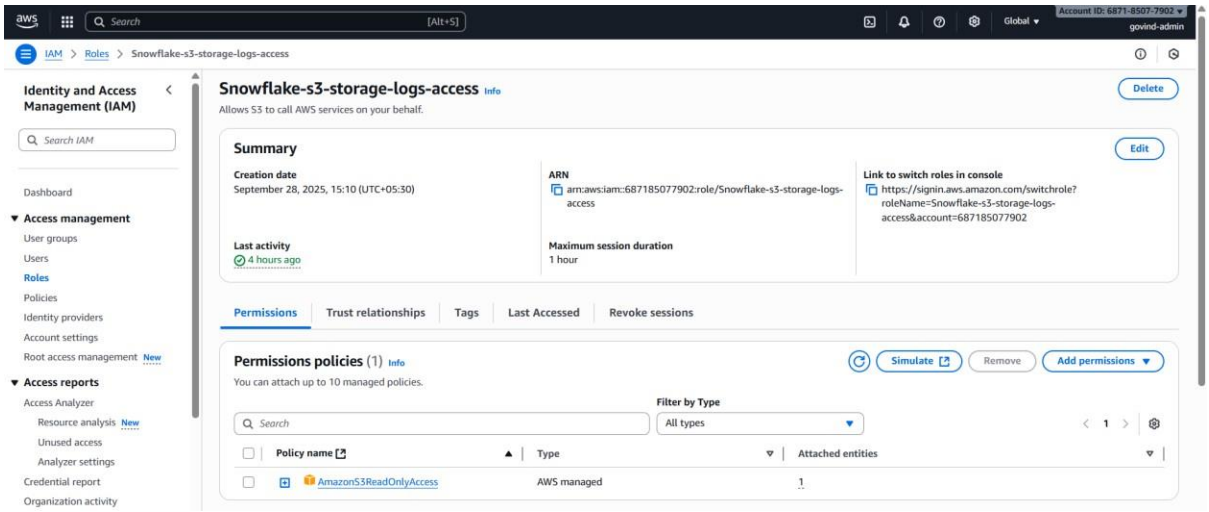
- ✓ Created a Warehouse (Sigmoid) , Database(logsdb) and Schema (RAW).



- ✓ Created a log_events Table to store the logs from S3.
- ✓ Create a JSON file format.

S3 READ POLICY(Role):

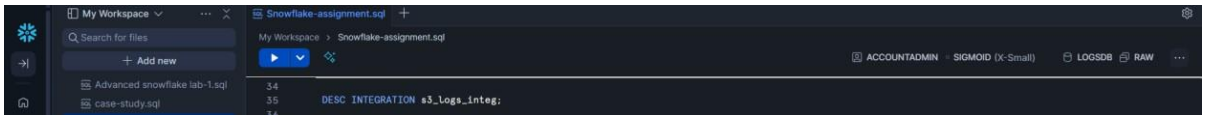
- ✓ Create a new role in IAM for S3 read access and assign to the Snowflake.



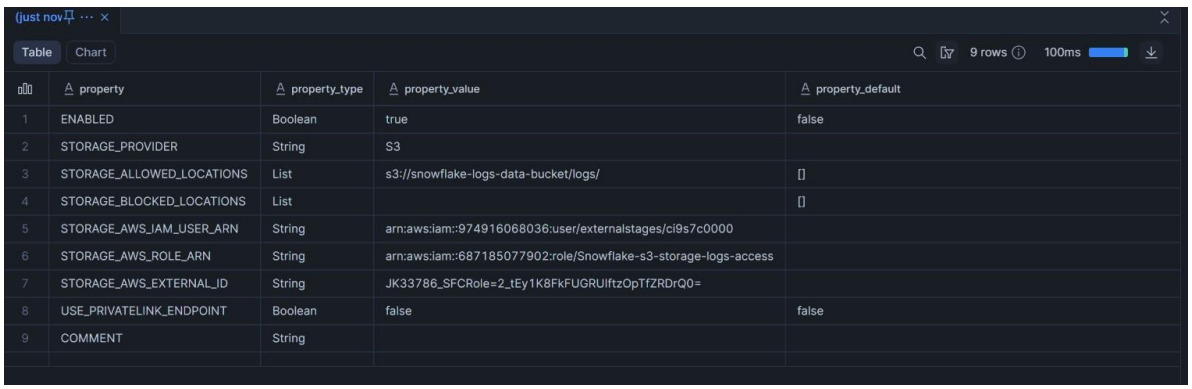
S3 and SNOWFLAKE INTEGRATION:

- ✓ Intregrate S3 and Snowflake, assign the role and the location of the bucket.

- ✓ Describe the created integration.

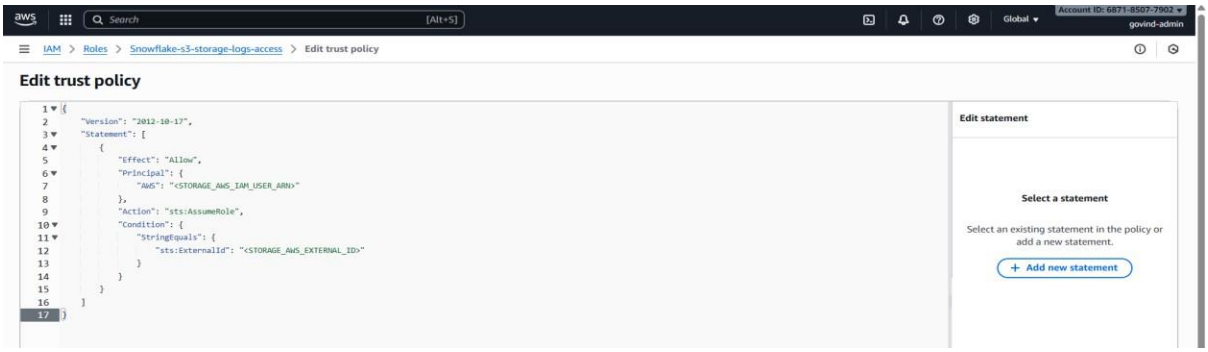


- ✓ Copy the ARN and External_ID from the OUTPUT.



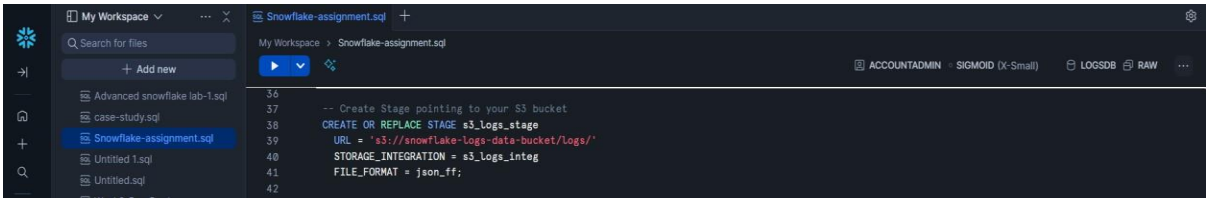
	property	property_type	property_value	property_default
1	ENABLED	Boolean	true	false
2	STORAGE_PROVIDER	String	S3	
3	STORAGE_ALLOWED_LOCATIONS	List	s3://snowflake-logs-data-bucket/logs/	[]
4	STORAGE_BLOCKED_LOCATIONS	List		[]
5	STORAGE_AWS_IAM_USER_ARN	String	arn:aws:iam::974916068036:user/externalstages/ci9s7c0000	
6	STORAGE_AWS_ROLE_ARN	String	arn:aws:iam::687185077902:role/Snowflake-s3-storage-logs-access	
7	STORAGE_AWS_EXTERNAL_ID	String	JK33786_SFCRole=2_tEy1K8FkFUGRUIftzOpTfZRDrQ0=	
8	USE_PRIVATELINK_ENDPOINT	Boolean	false	false
9	COMMENT	String		

✓ Update the Policy with ARN and External_ID in the Role policy.

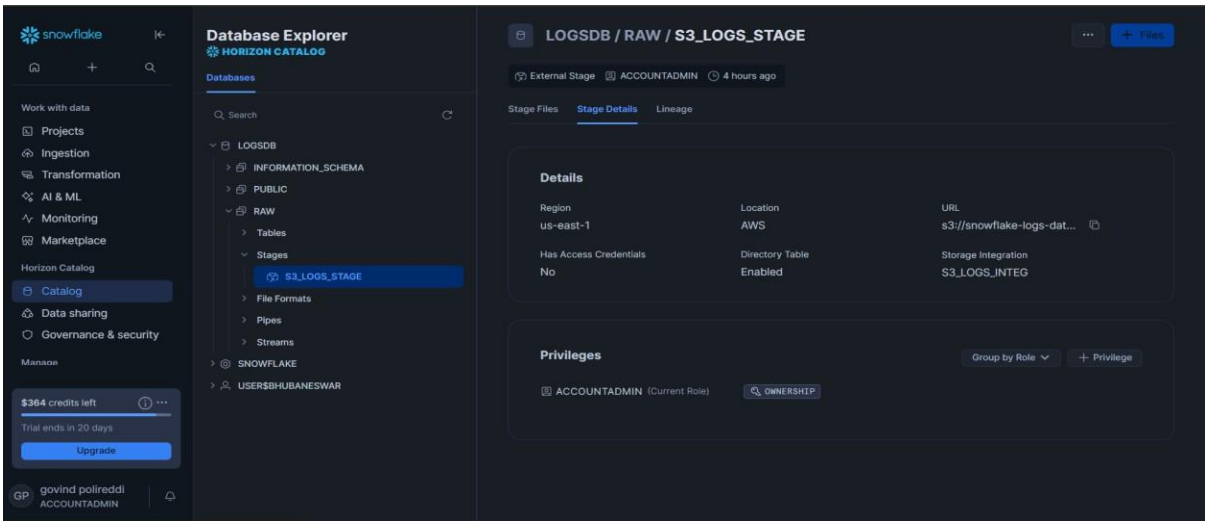


STAGE CREATION:

✓ Create a stage to point it to the S3 bucket to fetch the Json files.



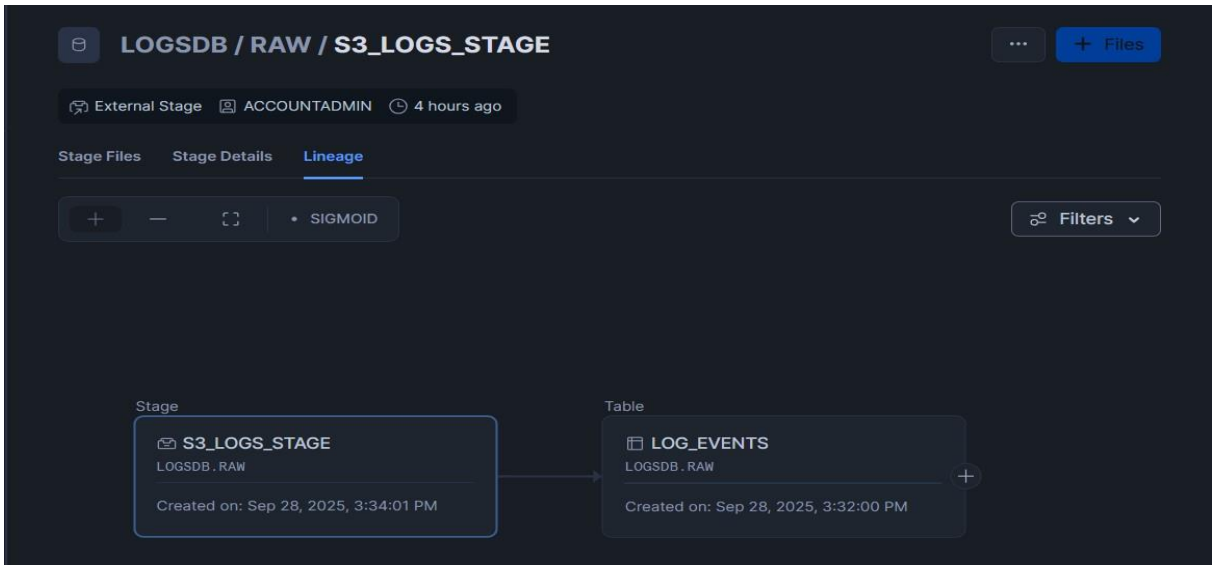
✓ Successfully created a stage.



SNOWPIPE:

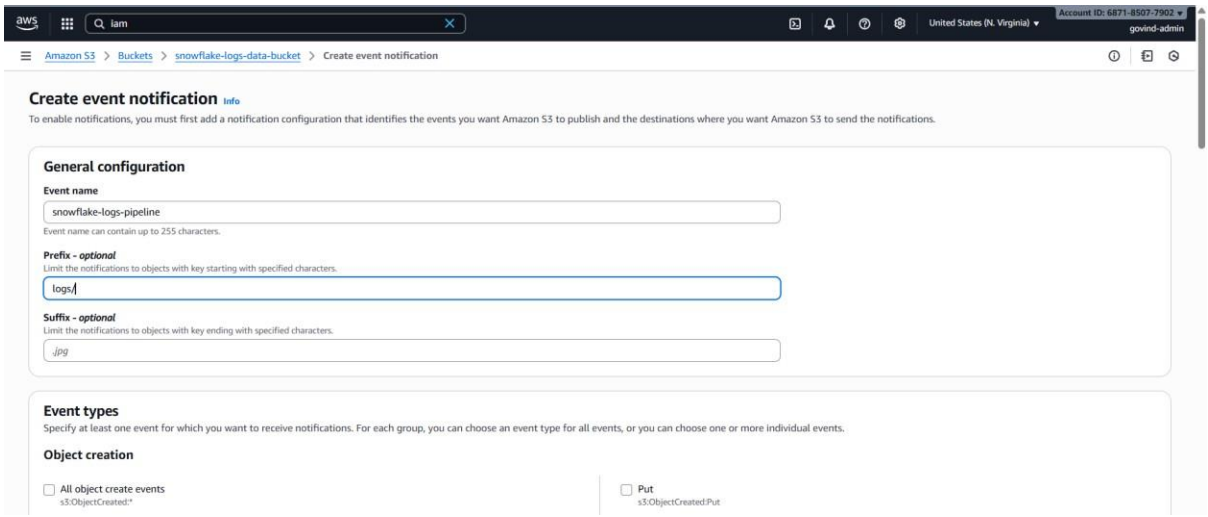
✓ Create a Snowpipe to ingest the data continuously into the logs table.

✓ Lineage graph is created from the Stage to the Table

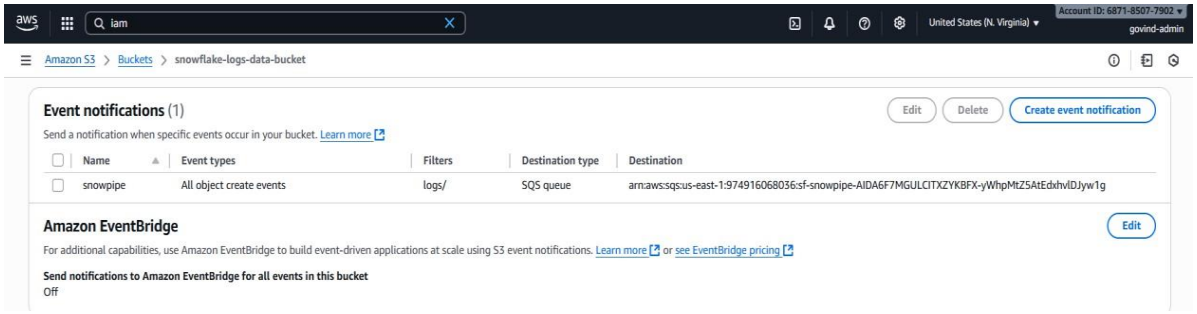


SQS EVENT NOTIFICATION:

✓ Create SQS Event notification ,when there is a new file added into the S3 bucket it gets trigger the Snowpipe.

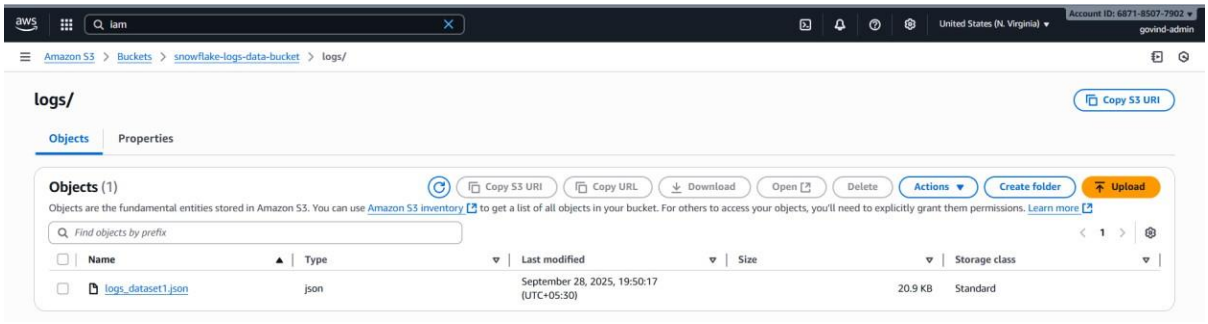


✓ Successfully created the SQS notification.

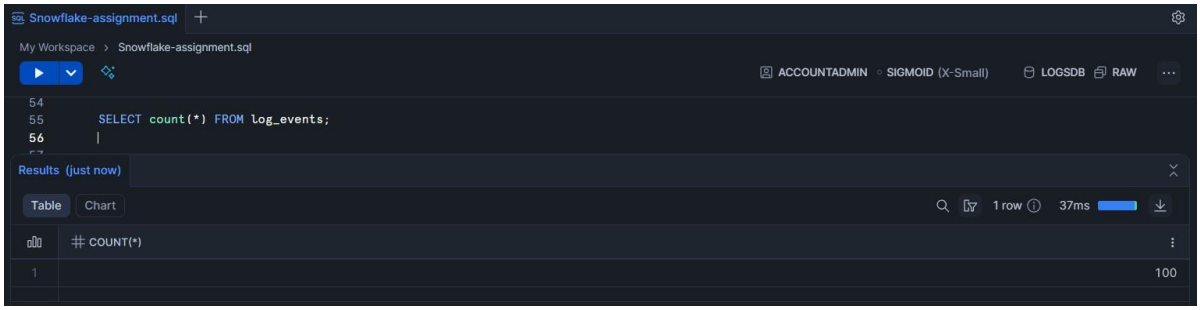


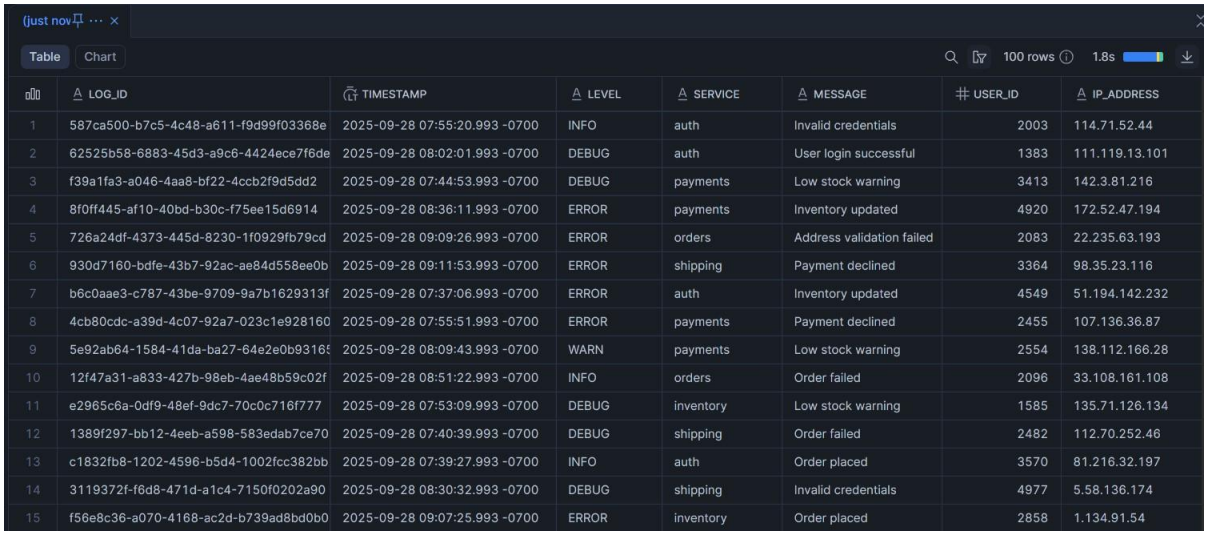
FILE UPLOAD:

- ✓ Uploaded a “logs_dataset1.json” file to the S3 bucket.

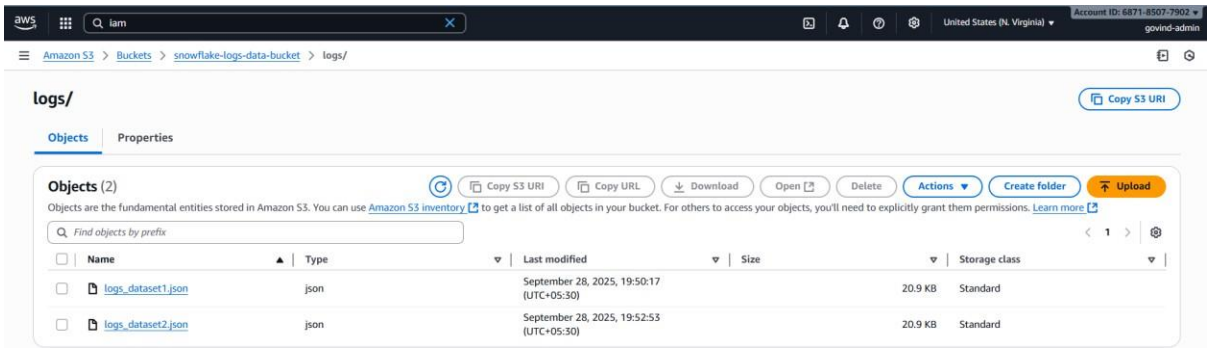


- ✓ Successfully loaded the logs data into the Table (100 Rows)

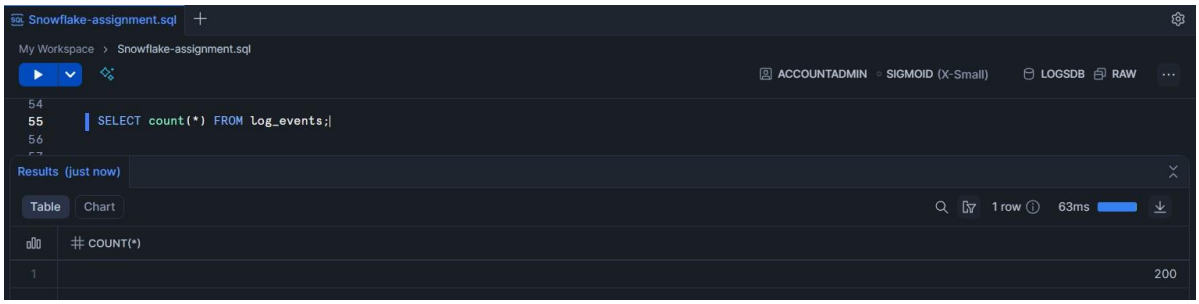


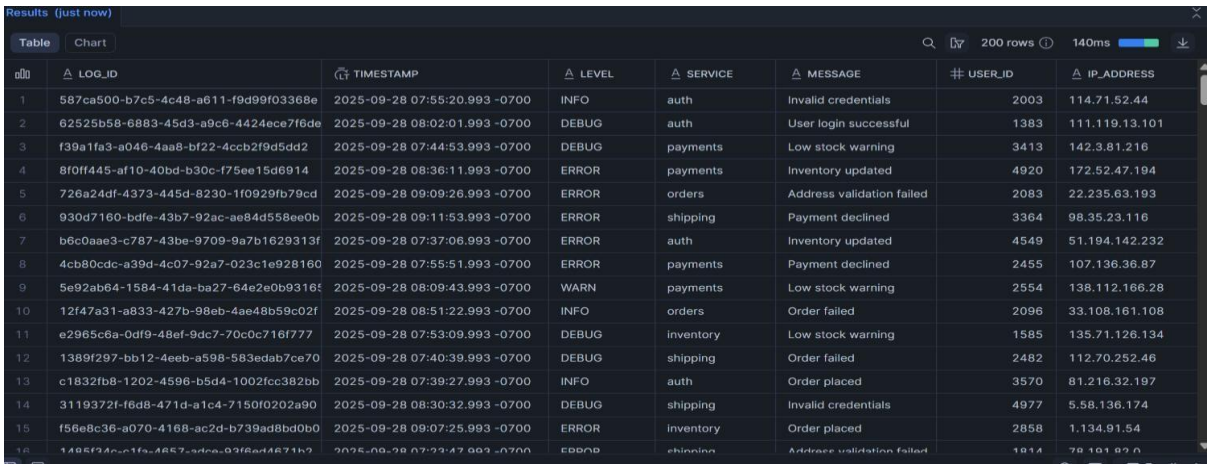


- ✓ Uploaded a “logs_dataset2.json” file to the S3 bucket.



✓ Successfully loaded the logs data into the Table (200 Rows)

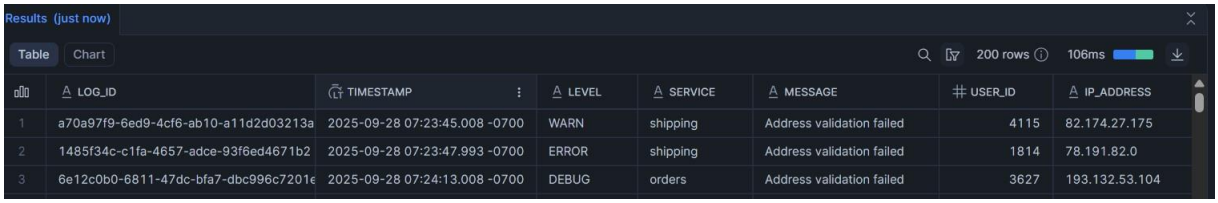




TIME TRAVEL:

✓ Retention period is 7 days.

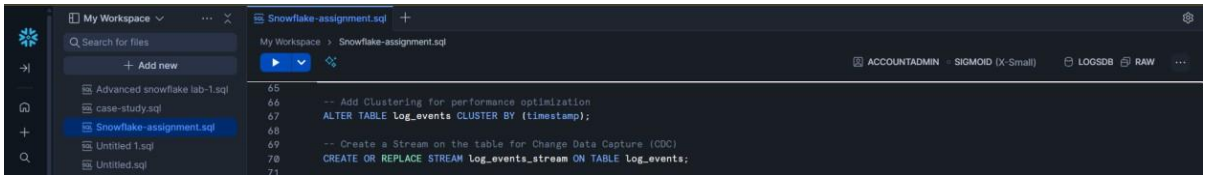
✓ Table logs data is of 200 Rows.(Present)



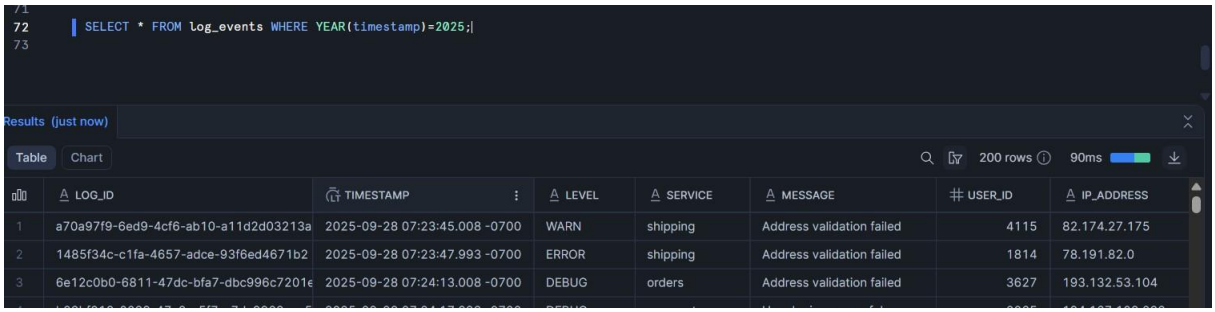
✓ Table logs data is of 100 Rows.(past of [400] seconds)

CACHE AND QUERY OPTIMIZATION:

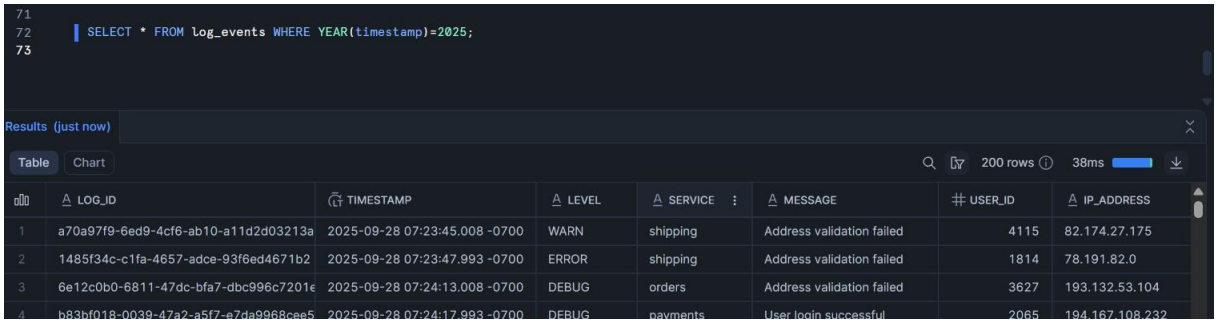
- ✓ Cluster the Column(timestamp) for faster optimization.



- ✓ 90ms time taken to retrieve the data as per the Query.

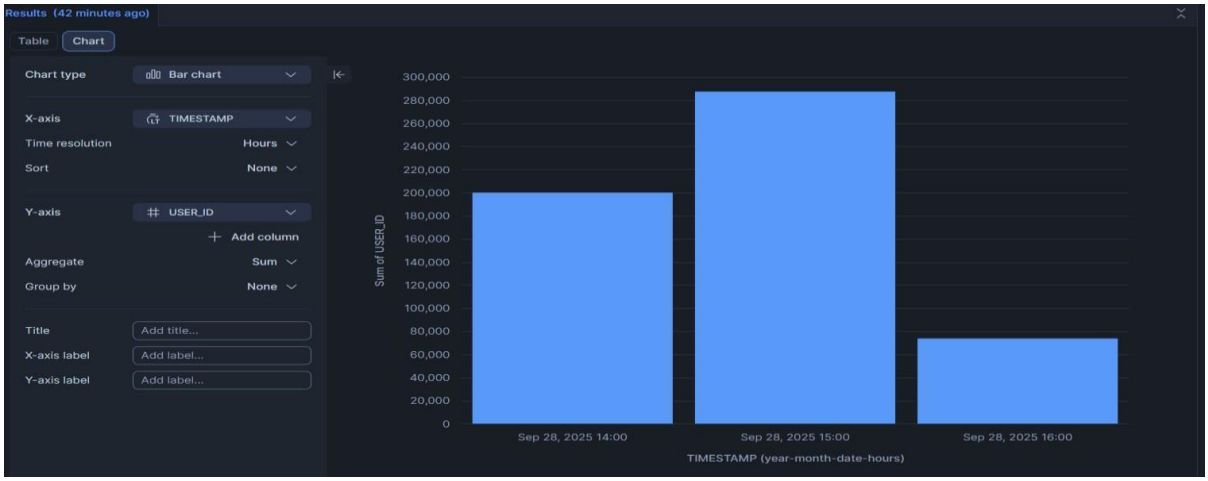


- ✓ 38ms time taken to retrieve the data as per the Query as it cached the Result(Result Cache).



MONITORING(SNOWSIGHT):

- ✓ Number of logs per day. 28th September got more logs.



✓ Error is having the main issue in the logs.

✓ Bar chart showing the message wise total logs.