Samples of Work

Nilson Peres

nilson.peres@usp.br, nilsontinassi2006@hotmail.com

1 Contextualization

The ultimate measure of an engineer is our work. As a Senior Support and Software Engineer at **T-Mobile**, I was responsible for supporting and enhancing a complex ETL pipeline—a system that I also helped design and implement.

This pipeline ingests data from multiple sources:

- APIs (MantaRay SON)
- XLSX files via SFTP

The data is transformed and loaded into several types of Snowflake tables:

- **Control Tables** – to orchestrate and validate execution.
- **Information Tables** – to hold metadata and enriched context.
- **Fact Tables** – both managed (permanent) and external.

For **external tables**, Snowflake maintains a link between the table and corresponding files stored in Azure Warehouse Storage. These tables require regular refreshes to ensure the latest data is visible.

The following section presents a representative support ticket where I combined technical depth, structured investigation, and a long-term fix.

2 Work Sample 1 – Snowflake Tables with Outdated Data

2.1 The Issue

A ticket was raised reporting that some Snowflake external tables were not displaying the latest data. This was unexpected, as the refresh mechanism for external tables had been working reliably for months.

2.2 Initial Investigation

To understand the problem, I first gathered context from the customer:

- When was the issue first noticed?
- Which tables were outdated?
- Was the issue global or limited to specific tables?

After collecting answers, I proceeded step by step:

1. **Checked file ingestion:** Using SFTP access to the Azure warehouse, I confirmed that new files were indeed being delivered.
2. **Validated refresh mechanism:** Snowflake external tables require refreshes. For cost and efficiency reasons, we had chosen not to enable auto-refresh, since thousands of files per second would cause excessive micro-transactions.
3. Instead, a scheduled task ran every 30 minutes, executing:
   `ALTER EXTERNAL TABLE <table_name> REFRESH`
4. **Reviewed task history:** I found repeated failures in the procedure execution.
5. **Manual testing:** Triggering the refresh manually showed that some tables updated successfully, but the process consistently failed on a specific table.

2.3 Root Cause Analysis

Snowflake's `ALTER EXTERNAL TABLE ... REFRESH` command re-registers all files between the external table and the warehouse. However, this operation is subject to internal limits on the number of files or total bytes that can be processed in one action.

Once the warehouse exceeded those limits, the refresh would fail systematically and indefinitely.

2.4 The Fix

I designed and implemented a more granular synchronization process:

- Instead of refreshing the entire warehouse at once, the procedure now refreshes by subfolder (daily partitions).
- Engineers can specify the number of days to refresh.
- A loop iterates through each day's folder, executing:

```
ALTER EXTERNAL TABLE my_external_table REFRESH 'folder_name/subfolder*'
```

This resolved the failure: all outdated external tables were successfully updated with the latest data.

2.5 Preventive Monitoring

Beyond fixing the issue, I focused on long-term resilience:

- Created a dashboard displaying the last update timestamp for each external table.
- Introduced thresholds and alerts for delayed refreshes.
- Enabled faster triage by support engineers, reducing mean time to detect (MTTD) and mean time to resolve (MTTR).

2.6 Key Takeaways

- **Ownership:** I handled the issue end-to-end, from customer communication to root cause analysis, fix design, and post-mortem improvements.
- **Balance of cost and performance:** The decision not to use auto-refresh was deliberate, showing awareness of both technical and business trade-offs.
- **Proactive mindset:** The monitoring solution turned a one-off issue into a permanently visible metric.

3 Work Sample 2 – Unsync Between Snowflake External Tables and Azure Warehouse

3.1 The Issue

A customer reported that some Snowflake external tables were **completely empty**. This was unusual, as the system had been stable for a long time, with most common issues already addressed.

3.2 Initial Investigation

As always, I began by gathering as much context as possible, either from the ticket or by asking the customer:

- When was the issue first observed?
- Did the tables ever contain data before, or were they always empty?

I validated the report myself:

1. Confirmed the reported tables were indeed empty.
2. Clarified with the customer: the tables **should have been capturing data for months**, but since there was no active use case, they had never been validated.
3. Checked GitLab for recent pipeline changes – none were relevant to this issue.

At this point, I focused on the three possible causes:

1. Missing source files in the Azure warehouse.
2. Incorrect linkage between Snowflake external tables and warehouse folders.
3. Sync failures during the refresh procedure.

3.3 Systematic Checks

- **(3) Sync procedure:** Triggered the refresh process manually, but no files were registered.
- **(1) Source files:** Connected to the Azure warehouse via SFTP (FileZilla). For multiple sample tables, I confirmed that files **were present** in the warehouse.
- **(2) Linkage:** Investigated the linkage between tables and warehouse folders. Here I discovered a critical detail: External table location reference used `UPPERCASE`, while the actual warehouse folder used `Standard_Name` format.

3.4 Root Cause

The mismatch came from a change in the *External Table Creator Service*:

- To simplify code, a global uppercase variable was introduced.
- This broke the link with the *Warehouse Manager Service*, which relied on the original naming convention.

As a result, external tables were created with links pointing to non-existent uppercase locations, leaving them permanently empty.

3.5 The Fix

To address the issue, I worked on three fronts:

(a) Code Fix

- Updated the *External Table Creator Service* to use standard names when creating links.
- Validated the fix by creating a new table: the link was generated correctly, and files were successfully ingested.

(b) Remediation of Existing Tables Two key challenges:

- Identify which external tables were affected.
- Map each external table to its correct warehouse folder.

Approach:

1. Wrote SQL queries joining:
   - All external table names.
   - Creation dates after the issue began.
   - Control tables (to check last processed files).

   This filtered the set of tables expected to have data but were empty.
2. To recover correct folder names, I wrote a shell script to connect via SSH to the warehouse and list all existing folders. This allowed a precise mapping between tables and folders.

(c) Automated Recreation Procedure

- Snowflake does not allow modifying an existing external table link.
- Designed a procedure to:
  1. Drop each incorrect external table.
  2. Recreate it with the correct folder mapping.
  3. Refresh to ingest historical files.

Outcome: About **1,500 external tables** were successfully remediated and populated with historical data.

3.6 Customer Communication

Delivered a detailed report explaining:

- Root cause and contributing factors.
- Exact fix and remediation process.
- Assurance that all historical data was restored.

3.7 Preventive Monitoring

As with previous incidents, I looked for ways to prevent recurrence:

- Created a new monitoring dashboard that continuously lists external tables expected to contain data but still empty.
- This gives support engineers a proactive tool: if a table shows up on this dashboard, it is a red flag requiring investigation.

3.8 Key Takeaways

- **Analytical approach:** Systematically eliminated possible causes until identifying the true issue.
- **Hands-on validation:** Used multiple tools (GitLab, SFTP, SQL, SSH) to confirm hypotheses.
- **Scalable fix:** Designed a bulk remediation solution covering 1,500+ tables.
- **Proactive mindset:** Turned a customer escalation into a permanent monitoring improvement.

4 Work Sample 3 – Provisioning a Feature (Stream Ingest V2) to a New Customer

4.1 The Context

Not all support tickets are about fixing bugs or investigating failures. Some are about **manual operations** or **feature enablement** for customers.

One of my specialties is leveraging my **software engineering background** to design automations and internal tools that save time, reduce errors, and improve consistency.

When I joined the support team, I quickly noticed a gap:

- Despite having five team members, **no automations or internal tools existed**.
- Every task—no matter how repetitive—was done manually.

Within my first month, I created three automations to address this gap. The **Stream Ingest V2 automation** was the first.

4.2 The Request

A new feature, *Stream Ingest V2*, had been added to our customer-facing framework.

- This was a complete redesign of the existing Stream Ingest solution.
- Provisioning required a deployment from scratch, involving multiple tasks such as editing the customer's Terraform configuration.
- Because this feature incurred additional cost, it was **only enabled upon customer request**, not by default.

The engineering team provided a runbook containing step-by-step manual instructions for support engineers to follow when enabling the feature.

While functional, this approach had several drawbacks:

- High manual effort and time consumption.
- Risk of human error.
- Inefficient use of support engineer bandwidth.

4.3 The Solution – Automation

Using the runbook as a blueprint, I developed an automation in Python:

1. Reads the Terraform file corresponding to the customer's cluster.
2. Checks the flags to determine if the feature is already active.
3. If not active, prompts the engineer for confirmation.
4. Automatically updates the Terraform configuration to add the necessary modules and flags.
5. Creates a pull request with the changes and submits it for approval.

This script replaced a long series of manual edits with a safe, repeatable process.

4.4 Example – Tooling Repository

The automation is available in my tooling repository:

`https://github.com/nilsinho42/sup-tools`

I later extended this framework to build similar tools for enabling other features, such as:

- **PrivateLink**
- **Scaling Fargate Pods**

4.5 Impact

- **Efficiency:** Reduced the time required to provision Stream Ingest V2 from a long manual process to an automated script execution.
- **Reliability:** Eliminated risk of human errors during Terraform edits.
- **Scalability:** Provided the foundation for automating future feature provisioning tasks.
- **Culture shift:** Demonstrated to the team the value of investing in support-focused engineering tools.

4.6 Key Takeaways

- **Proactive initiative:** Identified a gap in tooling just weeks after joining.
- **Engineering mindset:** Applied software development practices (automation, version control, pull requests) to improve support operations.
- **Lasting value:** Built a reusable automation framework that continues to help beyond the original request.