Snowflake Builders Blog: ...

Best practices to optimize data ingestion spend in Snowflake

Samartha Chandrashekar Follow 9 min read · Oct 30, 2023

124 1

This blog post reviews mechanisms to ingest data into Snowflake and then covers best practices to optimize costs incurred for data ingestion.

Data ingestion mechanisms in Snowflake

Snowflake supports ingesting data in multiple formats and compression methods at any file volume. Features such as schema detection and schema evolution simplify data loading directly into structured tables without needing to split, merge, or convert files.

First-party mechanisms for data ingestion include INSERT, COPY INTO, Snowpipe, Snowpipe Streaming, and Kafka Connector.

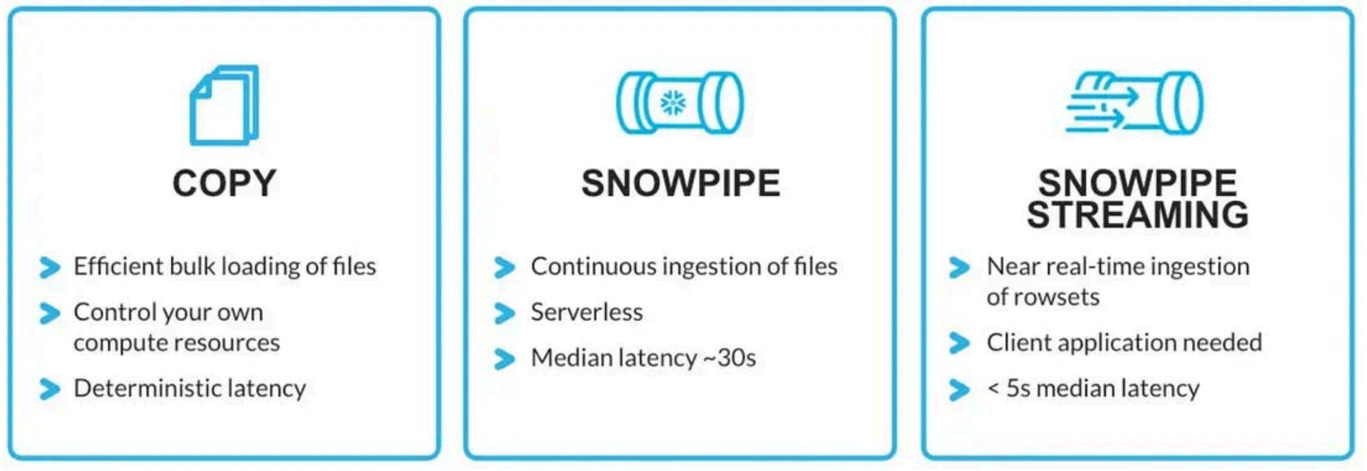
The INSERT command is the simplest ingestion mechanism and is suitable for bringing a small amount of data; however, it has scalability and error-handling limitations when dealing with data sets exceeding the single-digit MiB range.

COPY provides increased control but requires the customer to manage the compute (via settings such as warehouse size and job duration). It requires specifying the source of data files (in any data type including semi-structured formats) and a target table.

Snowpipe is a serverless service that enables loading data from files as soon as they're available in a Snowflake stage (locations where data files are stored for loading/unloading). With Snowpipe, data can be loaded from files in micro-batches rather than executing COPY statements on a schedule. Unlike COPY INTO which is a synchronous process that returns the load status, Snowpipe file ingestion is asynchronous and processing status needs to be observed explicitly. With Snowpipe, pricing is a function of the usage of Snowflake-managed compute and a per-file charge.

Snowpipe Streaming enables serverless low-latency streaming data ingestion directly into Snowflake tables (bypassing cloud object storage) with exactly-once and ordered delivery guarantees.

Additionally, the Snowflake Kafka Connector can ingest Kafka topics into Snowflake tables via Snowpipe.


COPY
Efficient bulk loading of files
Control your own compute resources
Deterministic latency
SNOWPIPE
Continuous ingestion of files
Serverless
Median latency ~30s
SNOWPIPE STREAMING
Near real-time ingestion of rowsets
Client application needed
< 5s median latency

Snowflake's provides batch and streaming ingestion options

Alongside first-party mechanisms, an extensive ecosystem of ETL/ELT tools and data ingestion partners can help move data into Snowflake.

Best practices to optimize ingestion costs

1/ Monitor the usage & cost of ingestion on an ongoing basis

For Snowpipes and COPY INTO, the COPY_HISTORY view or COPY_HISTORY function contains usage information. Snowpipe load history is recorded in the LOAD_HISTORY view or can be accessed via the COPY_HISTORY function. Similarly for Snowpipe streaming, load history is recorded in the SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY view.

The *pipe_usage_history* view can be used to monitor cost. The following query provides the full list of pipes and credits consumed over the last 30 days, broken out by day:

```
SELECT TO_DATE(start_time) AS date,
  pipe_name,
  SUM(credits_used) AS credits_used
FROM snowflake.account_usage.pipe_usage_history
WHERE start_time >= DATEADD(month,-1,CURRENT_TIMESTAMP())
```

```
GROUP BY 1,2
ORDER BY 3 DESC;
```

API-triggered Snowpipes can use insertReport (which returns events for the last 10 minutes) and loadHistoryScan (to track ingestion).

2/ Ensure that the number, size, and paths of files are aligned with the strengths of the ingestion modality

There is a fixed, per-file overhead charge for Snowpipe in addition to compute usage costs. As a result, for smaller single digit MiB or lower file sizes, Snowpipe can be less cost-effcient (in credits/TiB) than COPY depending on file arrival rate, size of warehouse, and non-COPY use of the Cloud Services Layer. Correspondingly, larger file sizes of at least 100 MiB are typically more efficient.

In general, we recommend file sizes above 10 MiB, with the 100 to 250 MiB range typically offering the best price-performance. As a result, we recommend aggregating smaller data files for batch ingestion. We also recommend not exceeding 5 GiB file sizes and splitting into smaller files to take advantage of parallelization. With a larger file, an erroneous record at the end may cause an ingestion job to fail and restart depending on the ON_ERROR option.

Finally, using the most explicit path allows COPY to list and load data without traversing the entire bucket, thereby saving compute and network usage.

3/ In general, prefer compressed files for ingestion

All of Snowflake's in-built ingestion mechanisms support common file formats such as CSV, JSON, PARQUET, AVRO, ORC, XML, etc. out of the box. If provided compressed, Snowflake decompresses them during ingestion.

Compression formats such as GZIP, BZ2, BROTLI, ZSTD, SNAPPY, DEFLATE, RAW_DEFLATE are detected automatically if not explicitly stipulated.

Directionally, the most significant observed impact on ingestion performance is file structure (i.e., number of columns or nested attributes in a single record) and not file size. Furthermore, file format, compression scheme, and nested structures impact compression efficiency. For example, an uncompressed file with many columns may take as long to load as a compressed file with a few columns but highly nested data. However, in general, ingestion of compressed files is preferred, as the movement of external assets across the network can be limiting compared to local decompression. In most cases, gzip compression and CSV format are the most performant for ingestion.

It usually is not beneficial to re-encode existing files (whose format is typically predefined by the source system), as the gain may not justify the cost. The only exception is during data loading across cloud regions or cloud providers, where greater compression can lower data transfer spend.

4/ Prefer Snowpipe over COPY INTO for most use cases

We recommend using Snowpipe for simplicity, convenience, and reduced management overhead.

The COPY command enables loading batches of data in cloud storage or an

to Snowflake tables.

Snowpipe provides a serverless experience with no virtual warehouses to manage; Snowflake manages and scales compute automatically while aiming for optimal price-performance based on the changing needs of the workload.

COPY provides file-level transaction granularity as partial data from a file will not be loaded by default. Snowpipe does not give such an assurance as it may commit a file in micro-batch chunks for improved latency and data availability. Snowpipe is designed for continuous ingestion; when loading data, a file is a chunking factor and not a transaction boundary determinant.

5/ Prefer using automated detection of new files for continuous loading with Snowpipe

To detect new files for ingestion with Snowpipe, either event notifications using cloud messaging services such as AWS SQS/SNS, Azure EventGrid/EventHuB, GCP Pub/Sub (that inform Snowpipe polling of the arrival of new data files) or Snowpipe REST endpoints can be used.

For most use cases with Snowpipe, automated ingestion with the former technique is preferred. This continuously loads new data to the target table by reacting to new files based on the parameters of the specified pipe object. Auto ingestion is also preferred when files have a steady arrival frequency while REST API calls are recommended when data arrives without a known frequency or if preprocessing is needed before ingestion.

REST API-triggered Snowpipe is suitable if an event service can’t be set up along with the requisite permissions or an existing data pipeline infrastructure is already in place or files are already present in a storage bucket before notification channels have been configured. It is also the only option if an internal stage is used for storing raw files. The REST API approach is the only option for ETL/ELT tools that don’t want to put the onus of managing object storage on the user and instead use Snowflake-managed internal stages.

Additionally, use Snowpipe error notifications for governance over failure alerting. Snowpipe error notifications can publish events to event handlers such as AWS SNS, Azure EventGrid, GCP Pub/Sub, etc.

6/ Right-size the virtual warehouses that back COPY jobs

COPY provides greater control, with the customer responsible for managing warehouse sizing/scaling and job duration based on the SLO/SLA needs of the use case. It is important to anticipate the memory and concurrency needs of the use case to right size compute. This includes not just warehouse size, type, and number of clusters but also settings for concurrency and timeout.

In cases of data skew, COPY jobs that don’t have enough files may not utilize the warehouse efficiently. For example, for loading less than 8 files, an XS warehouse (with MAX_CONCURRENCY_LEVEL = 8) will be just as fast as a 2XL. Therefore, it is important to consider the degree of parallelism and appropriately set the maximum concurrency level for the warehouse.

Long-running COPY jobs loading large numbers of files can get killed by job duration timeouts. This can be avoided by splitting into smaller jobs or changing the timeout.

Run multiple warehouse clusters concurrently to ingest separate tables in parallel. We suggest choosing the warehouse size based on the number of files, size, or format.

Ingestion jobs with complex UDFs can take significant time per row and can run out of memory if the data size is incorrectly anticipated. In such cases Snowpark-optimized warehouse with much higher memory:CPU ratios may be more helpful.

7/ Tune configurations of the Snowflake Kafka connector to optimize price-performance of ingestion with Kafka

Kafka (either self-hosted or managed) provides data collection and distribution infrastructure to write/read streams of records.

In addition to file size considerations and best practices that apply to other file-based ingestion mechanisms, while using Kafka, additional trade-offs between latency and larger file size can help with cost optimization.

Tuning Kafka with an awareness of the Kafka cluster's memory settings can help optimize price-performance for data ingestion.

The Kafka connector creates a file per partition per topic. Upon hitting buffer limits, the file is flushed and sent for ingestion through Snowpipe; subsequent offsets are buffered in memory.

The number of partitions (topics * avg partitions/topic) which may depend on other sinks and existing Kafka topic configuration, influences the number of files— more partitions result in multiple small files. We recommend minimizing partitions per topic unless there is a large message flow rate. We also recommend fewer Kafka partitions to the extent possible (especially if there is not a lot of data per minute in each partition).

Buffer.count.records, Buffer.flush.size, and *Buffer.flush.time* are settings in the Snowflake Kafka connector that influence the size & number of files per minute sent to Snowflake via the Kafka connector. The current defaults are: 1/ Buffer.count.records = 10000, 2/ Buffer.flush.time = 120 seconds, 3/ Buffer.flush.size = 5 MB

Lower latency via buffer.flush.time means smaller files and higher costs and vice versa. For example, doubling buffer.flush.time reduces the files/minute rate by a factor of 2. If Buffer.flush.size is increased, files/minute rate is reduced; reaching max buffer size earlier than max buffer time also has an impact.

Larger files lower cost by reducing the total number of files/TiB. This is because the Kafka connector uses Snowpipe whose pricing has a per-file charge.

Additionally, using the Avro format for Kafka messages makes it easier to leverage schema registry and schematization support.

8/ Use Snowpipe Streaming for low-latency use cases

For use cases that require low-latency ingestion and transformation with exactly-once or ordered delivery guarantees, we recommend using Snowpipe Streaming. Snowpipe Streaming enables data to be streamed directly into Snowflake tables. Snowpipe Streaming can be used via an API or using Snowflake-provided connectors for Kafka, MySQL, and Postgres. Additionally, partners such as like Fivetran, Informatica, Striim, etc. have also integrated Snowpipe Streaming to enable faster ingestion.

Snowpipe Streaming ingests rows via “channels” which represent streaming connections for loading data into a table. A single channel maps to exactly one table in Snowflake; multiple channels can also point to the same table. Behind the scenes, streamed data written to a target table is stored in a temporary intermediate file format. An automated, background process migrates data from active intermediate files to native files optimized for query and DML operations.

Unlike traditional Snowpipe which uses a pipe object to queue and loads staged file data into target tables, Snowpipe streaming doesn’t require a pipe object and writes records directly to target tables.

Snowpipe Streaming does not support TRANSIENT or TEMPORARY tables.

9/ Use Apache Iceberg where possible to reduce ingestion costs

Snowflake supports the Apache Iceberg file format that eliminates the need to move or copy tables between different systems, which can lower ingestion costs.

Iceberg tables managed by Snowflake offer similar performance to ingested Snowflake-format tables. Furthermore, Snowflake can be seamlessly be plugged into other Iceberg catalogs tracking table metadata. If your use case can tolerate the small performance difference across Iceberg tables and Snowflake-format tables, consider using Iceberg to save on data ingestion spend.

Conclusion and recap

Snowflake supports multiple native mechanisms to ingest data such as INSERT, COPY INTO, Snowpipe, Snowpipe Streaming, and Kafka Connector.

Our best practices for optimizing Snowflake data ingestion costs include: 1/ Monitor the usage & cost of ingestion on an ongoing basis. 2/ Ensure that the number, size, and paths of files are aligned with the strengths of the ingestion modality. 3/ In general, prefer compressed files for ingestion. 4/

Prefer Snowpipe over COPY INTO for most use cases. 5/ Prefer using automated detection of new files for continuous loading with Snowpipe. 6/ Right-size the virtual warehouses that back COPY jobs. 7/ Tune configurations of the Snowflake Kafka connector to optimize price-performance of ingestion with Kafka. 8/ Use Snowpipe Streaming for low-latency use cases. 9/ Use Apache Iceberg where possible to reduce ingestion costs.

Cost Optimization Snowflake Data Ingestion Data Engineering Data Pipeline

Published in Snowflake Builders Blog: Data Engineers, App Developers, AI, & Data Science

Following

10K followers · Last published 16 hours ago

Best practices, tips & tricks from Snowflake experts and community

Written by Samartha Chandrashekar

Follow

265 followers · 0 following

Product Managment at Snowflake

Responses (1)

Omar Essam he/him

What are your thoughts?

yugandhara saste

Jun 3

Question:

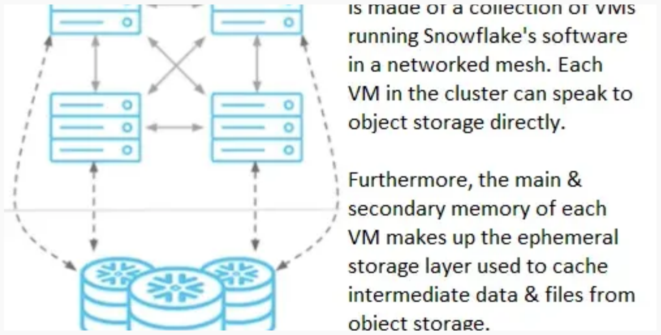
A Data Engineer needs to process daily transaction data from multiple tables across schemas, automatically including new tables, with error handling and efficiency prioritized. Which solution best meets these requirements?

Options:

A) Use… more

Reply

More from Samartha Chandrashekar and Snowflake Builders Blog: Data Engineers, App Developers, AI, & Data Science


is made of a collection of VMs running Snowflake's software in a networked mesh. Each VM in the cluster can speak to object storage directly.

Furthermore, the main & secondary memory of each VM makes up the ephemeral storage layer used to cache intermediate data & files from object storage.

In Snowflake Builders Blog: Data E… by Samartha…

Deep dive into the internals of Snowflake Virtual Warehouses

AGENTIC BUSINESS INSIGHT
Agent Apps Snowflake Intelligence
Agent APIs Cortex Agents
STATE OF THE ART RETRIEVAL
Unstructured Data Retrieval Cortex Search
Structured Data Retrieval Cortex Analyst
SCALABLE AI PROCESSING
Multimodal AI-powered SQL Cortex AISQL
Document Processing Document AI, Parse, Embed
MODELS
OpenAI
ANTHROPIC
Meta
deepseek
snowflake
GOVERNANCE
RBAC
Guardrails
Evaluations
Monitoring
AI Gateway
DATA
DATA INGESTION
ORACLE

In Snowflake Builders Blog: Data Engine… by Ume…

Build Snowflake Cost Savings and Performance Agent in 5 minutes

Snowflake's Data Cloud provided as Software-as-a-Service (SaaS), enables data storage,...

Sep 21, 2023 153 3

Snowflake Cortex Agent, building agentic AI solutions is no longer about complexity—it's...

Sep 30 44 3

In Snowflake Builders Blog: Data Enginee... by Sai...

Agent Instruction Best Practices for Snowflake Intelligence

Prototyping AI agents is easy. However, successfully launching reliable agents to...

Sep 25 54 3

ɔase Replication
eplication of data from one account to another
ınd and runs instantaneously
rough a Task
Database Failove
When one Snowflake account is unav
can be promoted to be primary. Thi
across cloud providers or re

In Snowflake Builders Blog: Data E... by Samartha...

Replication/failover in Snowflake and best practices to optimize...

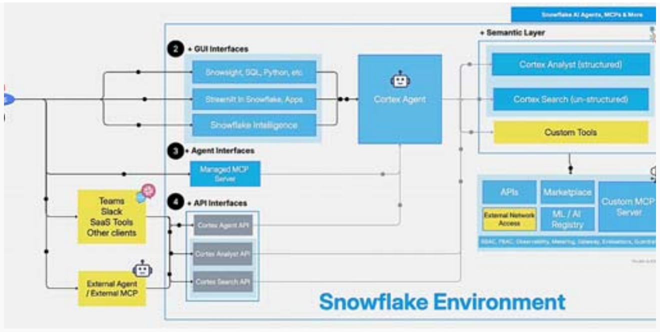
This blog post covers an overview of Snowflake functionality to ensure business...

Oct 24, 2023 7

See all from Samartha Chandrashekar

See all from Snowflake Builders Blog: Data Engineers, App Developers, AI, & Data Science

Recommended from Medium

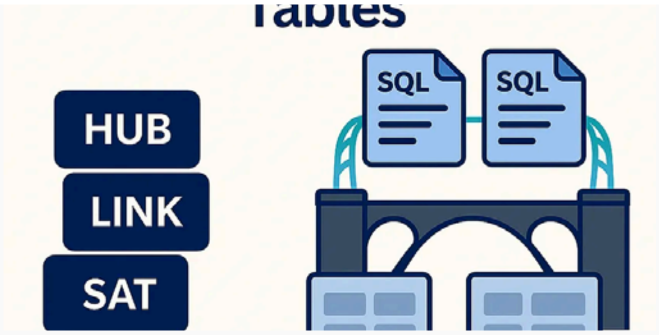


In Fru.dev by Fru

5 Strategic Pillars to Understanding Snowflake's AI...

A Complete Architecture Overview

Sep 16 23

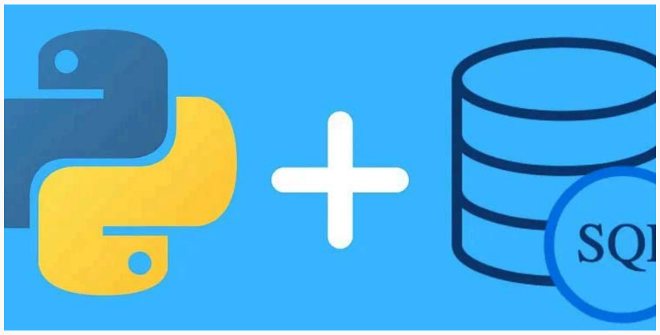


In Model Driven Data Enginee... by Jaco van der L...

Migrating from Data Vault to Bi-Temporal 3NF: A Controlled...

How SQL parsing, AI-assisted prompts, and structured validations enable safe migration...

Sep 21

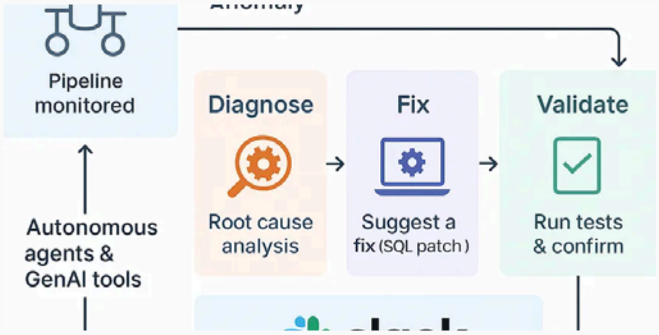


Sonal Singh

Snowflake Procedures: SQL vs JavaScript vs Python

When working with stored procedures in Snowflake, you have three language options:...

Sep 1



In Tech with Abhishek by Abhishek Kumar Gupta

🛠️ Building a Self-Healing Data Platform with GenAI &...

🧠 Introduction: From Alerts to Autonomous Repair

Jun 2 51

Pascal Pfäffle

Snowflake Openflow in Action: SQL Server CDC, API Integration, and...

Introduction

Jul 10 25 3

Laura Mitchell

Cutting Snowflake Costs: Monitoring Warehouse Costs

How to review warehouse uptime, idle gaps, runtime stats, and concurrency to decide if...

Oct 1 11 1

See more recommendations