SNOWPIPE

Imagine you have data arriving at an external stage every few minutes —**Snowpipe** automates the process of loading this new data into Snowflake as soon as it lands in a stage, keeping your data up to date without manual intervention.

Questions to Consider:

1. When should you use Snowpipe for data loading?
2. How does Snowpipe manage the compute resources automatically?
3. What are cost-effective ways to use Snowpipe for continuous data loading?

What is Continuous Data Loading?

If you need your data available in near real-time, **Snowpipe** is your go-to solution. **Snowpipe** automatically loads new data as it lands in a stage, keeping your database up-to-date without you having to run manual processes.

FEATURES:

- **Automatic Scaling**: Snowpipe automatically adjusts compute resources, meaning you don't have to worry about managing servers or scaling up during peak usage times.
- **Integration with Cloud Messaging**: Snowpipe can be triggered by cloud notifications (such as AWS SQS or Azure Event Grid) or via the Snowpipe REST API. This means Snowflake knows when new data is available and starts loading it right away.
- **Cost Considerations**: Snowpipe charges per file ingestion. We recommend files at least above 10 MB on average, with files in the 100 to 250 MB range offering the best cost-to-performance ratio.

Real-World Example

A media company collects real-time user engagement data. With Snowpipe, each time someone clicks on an article, the click data is stored in an S3 bucket and automatically loaded into Snowflake, allowing the company to track user behavior instantly.

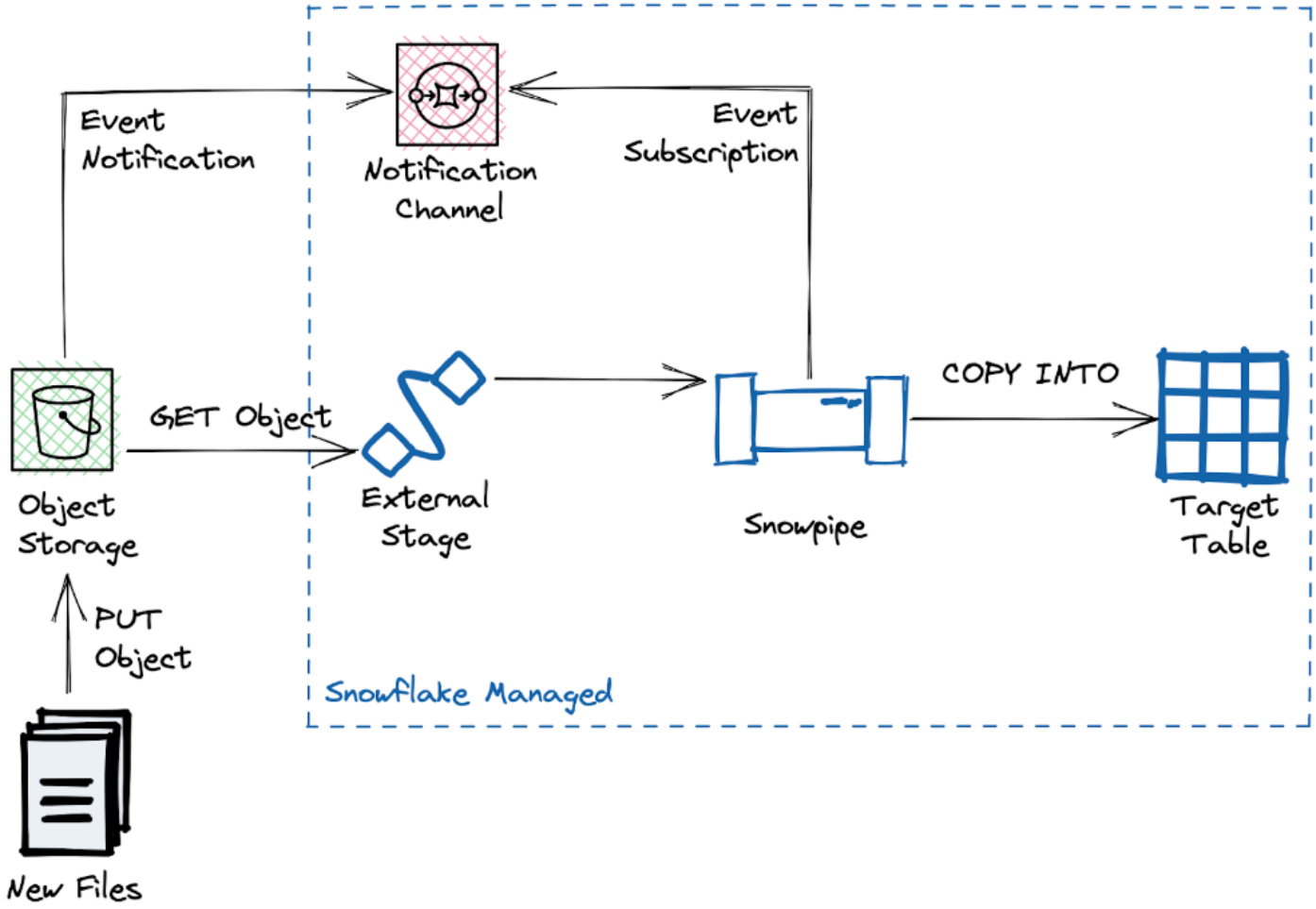
Common Pitfalls and Best Practices:

- **Pitfall**: Triggering Snowpipe for every tiny file can become costly.
- **Best Practice**: Group smaller files into larger batches to reduce the number of triggers and lower costs.

Best Practices for Data Ingestion with Snowflake

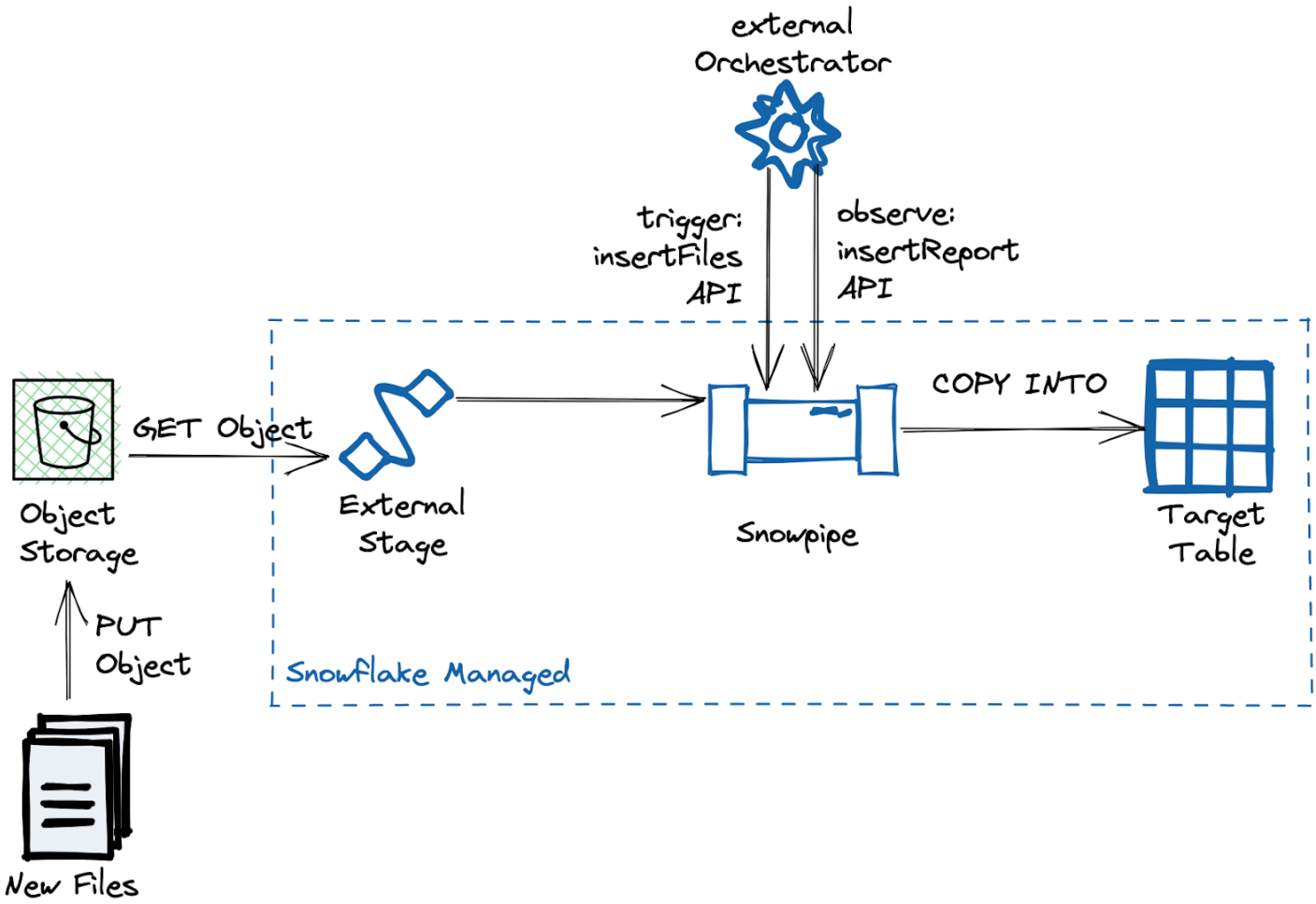
The Snowflake-provided serverless resources can operate for all operation types: batched, micro-batch, and continuous.

For most use cases, especially for incremental updating of data in Snowflake, auto-ingesting Snowpipe is the preferred approach. This approach continuously loads new data to the target table by reacting to newly created files in the source bucket.



Snowpipe relies on the cloud vendor-specific system for event distribution, such as AWS SQS or SNS, Azure Event Grid, or GCP Pub/Sub. This setup requires corresponding privileges to the cloud account to deliver event notifications from the source bucket to Snowpipe.

Whenever an event service can not be set up, or an existing data pipeline infrastructure is in place, a REST API-triggered Snowpipe is a suitable alternative. It is also currently the only option if an internal stage is used for storing the raw files. Most commonly, the REST API approach is used by ETL/ELT tools that don't want to put the burden of creating object storage on the end user and instead use a Snowflake-managed Internal Stage.



Here is an example of how to trigger a Snowpipe API for ingestion:
https://medium.com/snowflake/invoking-the-snowpipe-rest-api-from-postman-141070a55337

COPY INTO is a synchronous process which returns the load status as output. Whereas for Snowpipe, execution of the file ingestion is done asynchronously, so processing status needs to be observed explicitly.

For all Snowpipes as well as COPY INTO, the COPY_HISTORY view or the lower latency COPY_HISTORY function is available. Generally, files that fail to load need to be looked at by someone, so it is often good to check COPY_HISTORY less frequently than the typical file arrival rate. For all Snowpipes, error notifications are also available to publish events to your event handler of choice (AWS SNS, Azure Event Grid, or GCP Pub/Sub).

Additionally, for the API-triggered Snowpipes, you can use the insertReport and loadHistoryScan API endpoints to track the ingestion. While the insertReport endpoint returns events for the last 10 minutes only, this constraint can be configured using the loadHistoryScan endpoint. In any case, insertReport should be favored over the loadHistory as excessive usage of the latter tends to lead to API throttling.

Recommended file size for Snowpipe and cost considerations

There is a fixed, per-file overhead charge for Snowpipe in addition to the compute processing costs. We recommend files at least above 10 MB on average, with files in the 100 to 250 MB range offering the best cost-to-performance ratio.

At the single digit megabyte or lower range for average file size, Snowpipe is typically not the most cost-effective (in credits/TB) option. COPY may provide a better cost performance depending on the file arrival rate, size of warehouse used, and non-COPY use of the Cloud Services Layer. Ergo, there is no single correct answer below 10 MB, and further analysis is warranted. In general, larger file sizes of at least 100 MB are noticeably more efficient, such that an increase in file size does not change the credits/TB much. However, we also recommend not exceeding 5 GB in file size, to take advantage of parallelization and error handling capabilities. With a larger file, there is a higher likelihood of an erroneous record being found at the end of a large file that may cause an ingestion job to fail, which will later need to be restarted depending on the selected ON_ERROR option. Snowflake can and does handle much larger files, and customers have successfully loaded files larger in the TB range.

With these file size recommendations in mind, it should be noted that the per-file charge tends to be a small fraction of the overall cost.

File formats

All ingestion methods support the most common file formats out of the box:
CSV
JSON
PARQUET
AVRO
ORC
XML (currently in public preview)
Additionally, these files can be compressed, and Snowflake will decompress them during the ingestion process. Supported compression formats like GZIP, BZ2, BROTLI, ZSTD, SNAPPY, DEFLATE, or RAW_DEFLATE can be configured explicitly or detected automatically by Snowflake. In general, the ingestion of compressed files should always be preferred over uncompressed files, as the movement of external assets of the network also tends to become the limiting factor compared to local decompression.

Due to the large set of combinations, it’s not possible to predict the performance of the ingestion of a specific file format configuration exactly even given a file size. The most significant observed impact on data loading performance is the structure of the file (the number of columns or nested attributes in a single record), not the total file size.

With all that said, our measurement indicated that the ingestion of gzip’ed CSV files (.csv.gz) is not only the most widely used but usually the most performant configuration for ingestion. Further, the observations indicate the performance according to the file format listed above, starting with the fastest.

Most of the time the available file format and size is predefined by the source system or already available in object storage data. In this case, it’s not beneficial to split, merge, or re-encode existing files to a different format or size as the gain presumably will not pay back the efforts. The only notable exception for this is the loading of data across cloud regions or cloud vendors, where the transparent compression performed by Snowflake can make a massive difference in egress spending.

10 best practices

1. Consider auto-ingesting Snowpipe for continuous loading. See above for cases where it may be better to use COPY or the REST API.
2. Consider auto-ingest Snowpipe for initial loading as well. It may be best to use a combination of both COPY and Snowpipe to get your initial data in.
3. Use file sizes above 10 MB and preferably in the range of 100 MB to 250 MB; however, Snowflake can support any size file. Keeping files below a few GB is better to simplify error handling and avoid wasted work. This is not a hard limit and you can always use our error handling capabilities such as ON_ERROR = CONTINUE.
4. Keep max field size capped at 16 MB. Ingestion is bound by a Snowflake-wide field size limit of 16 MB.
5. Keep your data ingestion process simple by utilizing our native features to ingest your data as is, without splitting, merging, or converting files. Snowflake supports ingesting many different data formats and compression methods at any file volume. Features such as schema detection and schema evolution (currently in private preview) can help simplify data loading directly into structured tables.
6. Average measurements across customers are not likely to predict latency and cost. Measuring for a sample of your data is a much more reliable approach beyond the indicative numbers.
7. Do check file loading success/failure for Snowpipe using COPY_HISTORY or other options, such as subscribing to Snowpipe error notifications. Also occasionally check SYSTEM$PIPE_STATUS for the health of the Snowpipe.
8. Do not expect in-order loading with Snowpipe. Files are loaded concurrently and asynchronously in multiple chunks, so reordering is possible. If an order is essential, use event timestamp in data if possible, or use COPY and load sequentially.
9. Leverage your existing object path partitioning for COPY when possible. Using the most explicit path allows COPY to efficiently list and load your data as quickly as possible. Even though Snowflake can scalably list and load large volumes of data; you can avoid wasted compute and API calls by using path partitioning, especially when you already loaded the previous days, months, and years data which will just be ignored for deduplication.
10. Use cloud provider event filtering to reduce the amount of notification noise, ingestion latency, and costs from unwanted notifications with filtering on prefix or suffix events before it is sent to Snowpipe. Leverage the native cloud event filtering before using Snowpipe's more powerful regex pattern filtering.

1. In a load process, data arrives at these objects in what sequence?
   A. Pipe -> Stage -> Table
   B. Table -> Stage -> Pipe
   C. Stage -> Pipe -> Table
   D. Table -> Pipe -> Stage
   E. Stage -> Table -> Pipe

2. When deciding whether to use bulk loading or Snowpipe, which factors should you consider? Select three.
   A. How often you need to load the data.
   B. Location of data (local system or cloud).
   C. Data format (structured or semi-structured).
   D. Number of files you will load at one time.

3. Snowflake supports landing data into which two locations? Select two.
   A. Internal stage on cloud storage platform.
   B. Internal stage on your local system.
   C. External stage on cloud storage platform.

4. When loading data in Snowflake, which two statements are true?
   A. All external cloud storage services charge the same price for data transfer.
   B. Snowflake supports loading CSV, JSON, Parquet, Avro, and ORC.
   C. The COPY INTO command cannot be used to load single files.
   D. Snowsight provides an interface for loading data into tables without having to write any code.

5. Who can provide the compute resources used by Snowflake for data-loading jobs? Select two.
   A. User-managed virtual warehouse.
   B. Hardware provisioned by user directly from cloud providers.
   C. Snowflake-managed serverless compute.

ANSWER KEYS

1. C
2. ABD
3. AC
4. BD
5. A

HAPPY LEARNING
REGARDS
SARANSH JAIN