Setting up a data pipeline using Snowflake's Snowpipes in '10 Easy Steps'

- **1) Set up a separate database**
- **2) Set up a schema to hold our source data**
- **3) Create a Table**
- **4) Create the File Format**
- **5) Create an external stage pointing to your s3 location**
- **6) Review staged files and select data from the files**
- **7) Test loading data into the table**
- **8) Create the Snowpipe**
- **9) Force a pipe refresh**
- **10) Monitor data loads**

We look at the steps required to set up a data pipeline to ingest text based data files stored on **s3** into Snowflake using **Snowpipes**.

Snowpipes is one of the more unique and powerful, yet somewhat under-documented, or at least not much talked about features in Snowflake.

1) Set up a separate database

Permalink

WIe like to set up a separate **database** in Snowflake for any source datasets that don't come in via Fivetran. (Also, I keep all source data outside our analytics database to begin with, usually in a database called `raw` or something similar.)

For example, let's create a database called `etl`:

```
create database etl;
```

```
use etl;
```

2) Set up a schema to hold our source data

Permalink

For the sake of this example, we'll load our data in the `src` schema:

```
create schema src;
```

3) Create a Table

Permalink

Since we don't have the benefit of **Fivetran** creating our table, we need to create a **table** to hold our data.

Let's assume a fictional table called `my_source_table` with 2 columns:

```
create table src.my_source_table
(
        col_1 varchar,
        col_2 varchar
);
```

4) Create the File Format

Permalink

The first real step is to create a `file format` that lets us control the type of file we want to ingest.

Even if your data is in a simple **csv** file, it makes sense to explicitly control the file format options. Snowflake provides a host of file format options here.

For our example, we create a comma-delimited file format with a header that has values quoted in `"` and may have `null` fields encoded with the string `'null'`. (I mean, really, who does that?)

```
create or replace file format my_csv_format
  type = csv field_delimiter = ',' skip_header = 1
  field_optionally_enclosed_by = '"'
  null_if = ('NULL', 'null')
  empty_field_as_null = true;
```

Let's check the existing file formats to make sure this got created:

```
show file formats;
```

5) Create an external stage pointing to your s3 location

Permalink

A `stage`, in Snowflake terms, is a pointer to an internal or external (i.e. *your*) s3 location where your data files are stored - encoding not just the path, but also any authentication information. (More here.)

```
create or replace stage my_stage url='s3://my_bucket/key/key/'
  credentials=(aws_key_id='KEY' aws_secret_key='SECRET')
  file_format = my_csv_format;
```

Or use an IAM role:

```
create or replace stage my_stage url='s3://my_bucket/key/key/'
  credentials=(aws_role='aws_iam_role=arn:aws:iam::XXXXXXX:role/XXXX')
  file_format = my_csv_format;
```

Let's review all of our stages in this database:

```
show stages;
```

6) Review staged files and select data from the files

Permalink

Let's make sure security and file formats are both working as expected. The best way is to use the `list` command to get a listing of files in our staging location.

```
list @my_stage;
```

You can also query the raw files directly to make sure the delimiters are working as expected, although we don't recommend using this as anything but a way to debug issues, and definitely not to read data files for production purposes. (More here.)

```
select t.$1, t.$2
from @my_stage (file_format => my_csv_format) t;
```

7) Test loading data into the table

Permalink

Before we set up a **Snowpipe**, we should make sure we can actually import data from the files into the table we've set up. Snowflake provides a few ways to limit the number of files we can copy to our table, which is especially helpful during testing if you have a lot of files.

For example, you can use a RegEx `pattern` to limit the number of files to load. (More here.)

```
copy into src.my_source_table
  from @my_stage
  file_format = my_csv_format
  pattern='.*sales.*.csv';
;
```

Since Snowpipes by default only load data staged in the last 7 days, it makes sense to load all files at this point if you have a lot of history to load. We can then use the Snowpipe to incrementally load new files.

Snowflake provides a number of error handling options for the `COPY INTO` command that are worth reviewing. For example, we could instruct Snowflake to `continue` loading rows when it encounters errors, or to skip the entire file if it encounters errors loading any rows.

```
copy into src.my_source_table
    from @my_stage
    file_format = my_csv_format
    on_error='continue'
```

Depending on the option set, the output of this command will provide detailed error information.

8) Create the Snowpipe

Permalink

Ultimately, we want to automate the copy command so we need a Snowpipe that will make this a bit easier to manage. (More info on Snowpipes.)

```
create pipe if not exists my_pipe as
copy into src.my_source_table from @my_stage;
```

Confirm that this worked as expected:

```
show pipes;
```

9) Force a pipe refresh

Permalink

Using the `alter pipe` command and the `refresh` option we can force a **Snowpipe** to send any files from its associated stage to an ingestion queue. You can read more here.

```
alter pipe my_pipe refresh;
```

This simple command allows you to force Snowflake to read the staged files and import them in the table specified in the pipe setup. If you have a way to automate the execution of simple SQL command (e.g. via dbt) then you can automate this!

Since this sends files to a queue, we'll wait a bit for Snowflake to process the queue of staged files, then we'll verify your post-load row row count.

10) Monitor data loads

Permalink

We can use the built-in `pipe_status` command to check on our pipe's status and how many files are current in the queue.

```
select system$pipe_status('my_pipe');
```

If files are no longer queued, check to make sure your table has the expected number of records.

```
select count(*) from src.my_source_table;
```

Snowflake provides a couple of ways to check on load success and/or errors.

The `COPY_HISTORY` function provides useful information of load status by file.

```
select *
from table(information_schema.copy_history(table_name=>'MY_SOURCE_TABLE',
  start_time=> dateadd(hours, -24, current_timestamp()));
```

In my experience this approach often doesn't yield any results. So, if you have `ACCOUNTADMIN` access, you can also query the

equivalent view in the ACCOUNTUSAGE schema directly, and also aggregate it to provide a status overview as shown below.

```
use role accountadmin;
use snowflake;

select
    convert_timezone('America/Los_Angeles', h.last_load_time)::timestamp_ntz::date as
load_date,
    max(convert_timezone('America/Los_Angeles', h.last_load_time)::timestamp_ntz) as
max_load_time,
    sum(h.row_count) as rows_loaded,
    sum(h.error_count) as errors
from account_usage.copy_history h
where table_name = 'MY_SOURCE_TABLE'
group by 1
order by 1;
```

Hopefully this has given you some insights into using Snowpipes for data pipelines that can’t be handled by your favorite data pipeline SaaS vendor.