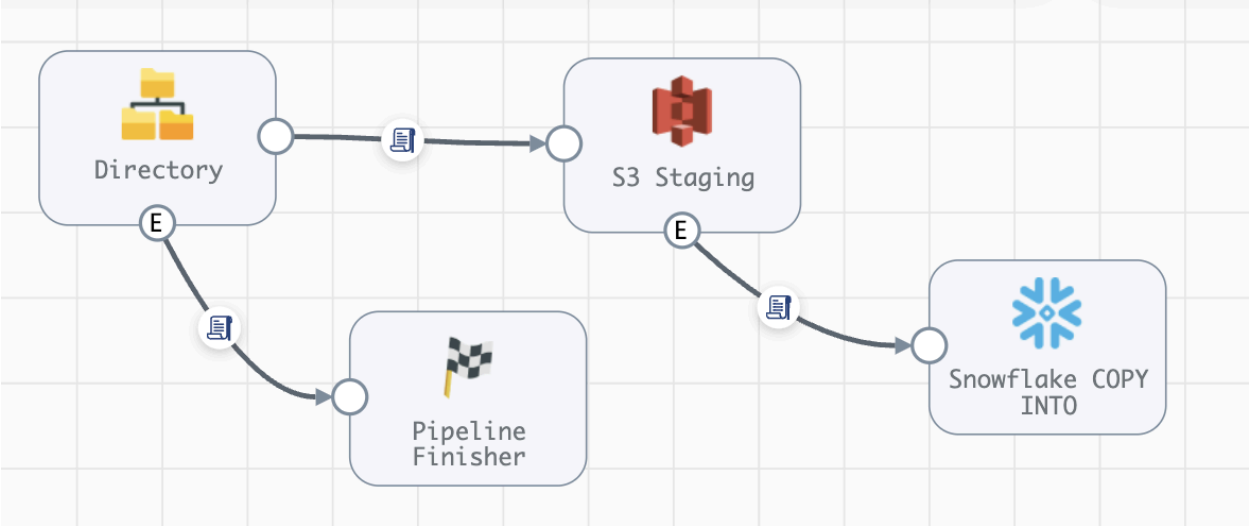
REST API to Snowflake Using curl and COPY INTO

This document describes an example pipeline that performs the following steps:

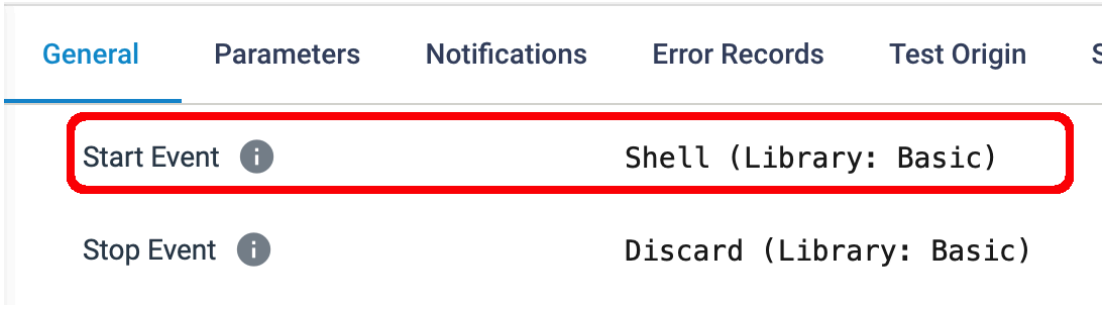
1. Calls a REST API to retrieve an arbitrarily large payload and writes the payload to a file.. The REST API is implemented using the microservice posted here.
2. Trims the first 4 and last 2 lines from the file
3. Moves the file to an S3 external stage
4. Executes a COPY INTO command to load the file into Snowflake
5. Executes any additional Snowflake commands

The pipeline can be downloaded from here.

The pipeline looks like this:



Steps 1 and 2 of the pipeline's actions (from the list above) are performed by a Shell Executor Start Event:



The Start Event Shell Executor is configured with this script

```
#!/usr/bin/env bash

# create a unique file name
FILE_NAME=data-$(date +"%s")

# Call the REST API and write the response to the file
curl -X GET  $REST_API_URL  -H "Content-Type:application/json" -o
$FILE_WORKING_DIR/$FILE_NAME

# Replace \" with " and \n with newlines
sed -i 's/\\\"/"/g;s/\\n/\n/g' $FILE_WORKING_DIR/$FILE_NAME

# Trim the first 4 and last 2 lines
cat $FILE_WORKING_DIR/$FILE_NAME | sed '1,4d;s/"sobjects"://' | head
-n -2 >> $FILE_WORKING_DIR/$FILE_NAME.ready

# Move the file to the pipeline's pickup dir
mv $FILE_WORKING_DIR/$FILE_NAME.ready
$FILE_PICKUP_DIR/$FILE_NAME.$FILE_PICKUP_EXTENSION

exit 0
```

The script will run when the pipeline is first started, and will move a file to the "pickup" directory when it is done. See the example pipeline's parameter list for example parameter values.

The pipeline's directory origin will pick up the file from the "pickup" directory and use Whole File Data Format to efficiently move the file to the S3 external stage.

When the file transfer is complete, an event is fired which triggers the Snowflake Executor that executes a COPY INTO command, and optionally, any additional commands.

One could also configure a pipeline Stop Event to execute any additional Snowflake commands when the pipeline is complete.

Important Notes

- For production use of the Shell Executor, one should configure Shell Impersonation Mode.

- The file that is moved to the staging area in this example and copied into Snowflake has comma delimited JSON records, like this:

```
{
  "activateable": false,
  "associateEntityType": null,
  ...
},
{
  "activateable": false,
  "associateEntityType": "ChangeEvent",
  ...
},
{
  "activateable": true,
  "associateEntityType": "ChangeEvent",
  ...
}
```

  This file format allows Snowflake COPY INTO commands to load the JSON records into Snowflake tables with a single variant column.

  If one wants to take advantage of Data Collectors Snowflake Destination, which could write this data to strongly typed, multi-column tables (with optional auto-table creation) , with data-drift handling and record-level metrics, the JSON records would need to be newline delimited, rather than comma delimited.

For example, like this:

```
{
  "activateable": false,
  "associateEntityType": null,
  ...
}
{
  "activateable": false,
  "associateEntityType": "ChangeEvent",
  ...
}
{
  "activateable": true,
  "associateEntityType": "ChangeEvent",
  ...
}
```

The conversion from comma delimited to newline delimited could be implemented either in the Start Event's shell script, or in a separate pipeline.

This approach would also allow record level enrichment within the pipeline, for example adding a timestamp to each record before being written to Snowflake