

SDC Pipeline Example: SFTP to S3 External Stage to Snowflake

Introduction	2
Prerequisites	2
Example Pipeline Download Link	2
Import the Pipeline into your environment	3
Set the Pipeline's Parameters	5
Set an Error Record Location	6
SFTP Origin Stage	7
S3 Staging Stage	8
Get S3 File Name Stage	10
Copy File into Snowflake	11
Validate the Pipeline	12
Preview the Pipeline	13
Run the Pipeline	15

Introduction

This document describes an example Data Collector pipeline that retrieves files from an SFTP site using [Whole File Data Format](#), copies the files to an S3-based external staging directory, and then executes COPY INTO commands to load the files' records into a Snowflake table.

All artifacts are available in this GitHub repo:

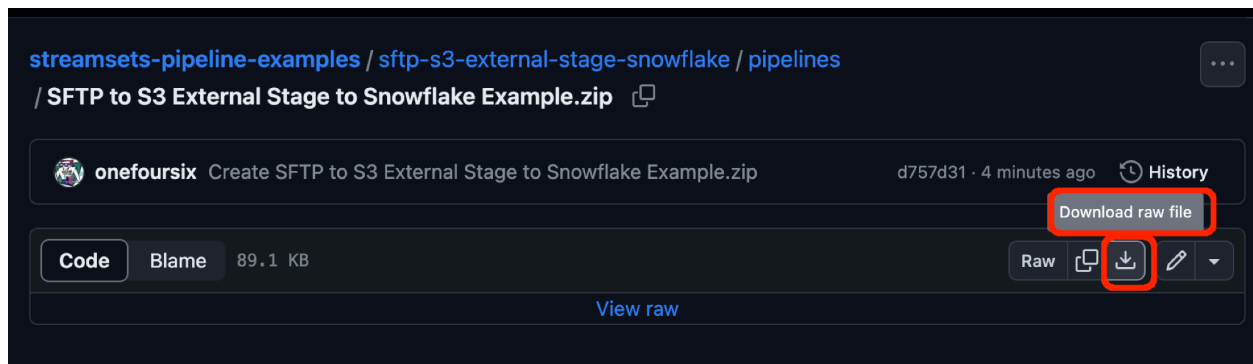
<https://github.com/onefoursix/streamsets-pipeline-examples>

Prerequisites

- The target Snowflake table must exist in advance
- The S3-based External Stage must exist in advance
- A Snowflake File Format must exist that correctly parses the source files

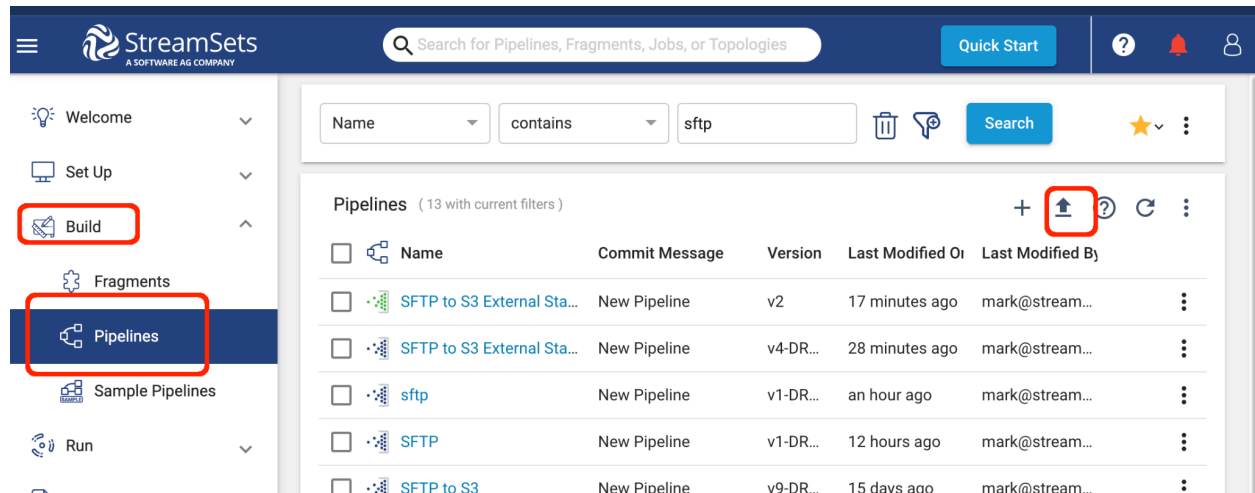
Example Pipeline Download Link

The pipeline can be downloaded (as a zip file) using the download button on [this page](#):



Import the Pipeline into your environment

Use the upload button in the Pipeline List View to import the pipeline:



Browse to the download zip file, and import it as a Pipeline Archive:

Import Pipeline

1 Upload File

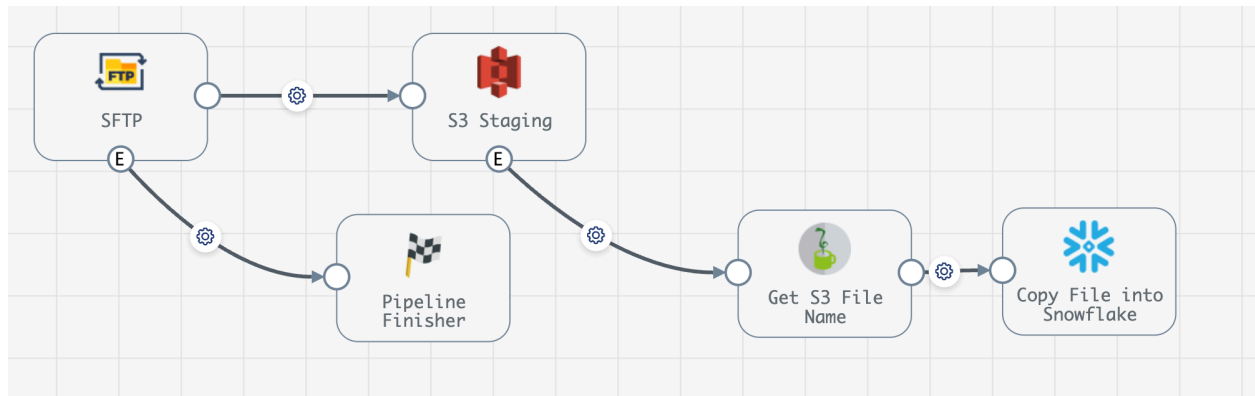
Select the import file type and enter a commit message for the imported pipeline versions. Then select the file containing the pipelines to import. [Learn More](#)

Import File Type: ☒ Archive File ☐ Pipeline JSON

Commit Message: *

SFTP to S3 External Stage to Snowflake Example.zip

The imported pipeline should look like this:



Set the Pipeline's Parameters

Edit the imported pipeline and set the pipeline's parameters to fit your environment. The example pipeline has these sample settings from my own environment:

GeneralParametersNotificationsError RecordsAdvancedError RecordsTest Origin

Hide Advanced Options ^

Parameters

SFTP_BASE_URL	:	sftp://35.185.193.227:22/files	-
SFTP_FILE_PATTERN	:	*.csv	-
SNOWFLAKE_WH	:	MARK_WH	-
TARGET_DB	:	MARK_DB	-
TARGET_SCHEMA	:	MARK_SCHEMA	-
TARGET_TABLE	:	EMPLOYEE	-
STAGE_DB	:	MARK_DB	-
STAGE_SCHEMA	:	MARK_SCHEMA	-
STAGE_NAME	:	MARK_EXTERNAL_STAGE	-
FILE_FORMAT_DB	:	MARK_DB	-
FILE_FORMAT_SCHEMA	:	MARK_SCHEMA	-
FILE_FORMAT_NAME	:	QUOTED_CSV_FORMAT	-
S3_STAGE_BUCKET	:	146bucket	-
S3_STAGE_BASE_DIR	:	snowflake_external_stage	-

The use of each parameter will be shown in the steps that follow.

Set an Error Record Location

Set the error record location to S3:

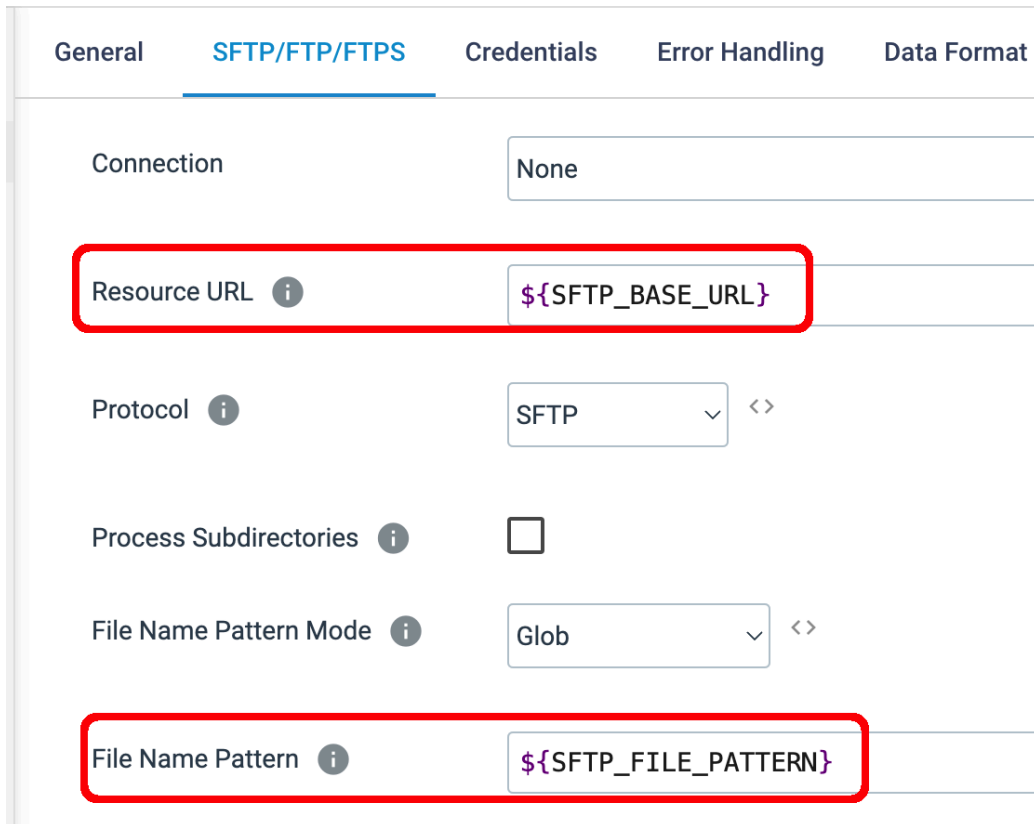
The screenshot shows the 'Error Records' tab selected in a configuration interface. The tab is highlighted with a red box. Below the tab, there are two sections: 'Error Records' and 'Error Record Policy'. The 'Error Records' section has a dropdown menu set to 'Write to Amazon S3 (Library: Amazon Web Services)', which is also highlighted with a red box. The 'Error Record Policy' section has a dropdown menu set to 'Original record as it was generated by the origin.'.

Set an S3 Bucket and folder to hold error records:

The screenshot shows the 'Error Records' tab selected in a configuration interface. The tab is highlighted with a red box. Below the tab, there are several settings for Amazon S3. The 'Connection' is set to 'None'. The 'Authentication Method' is set to 'Instance Profile'. There are checkboxes for 'Assume Role' and 'Use Specific Region', both of which are unchecked. The 'Bucket' field is set to '<YOUR BUCKET FOR STREAMSETS ERRORS>' and the 'Common Prefix' field is set to '<YOUR FOLDER FOR STREAMSETS ERRORS>'. Both the 'Bucket' and 'Common Prefix' fields are highlighted with a red box.

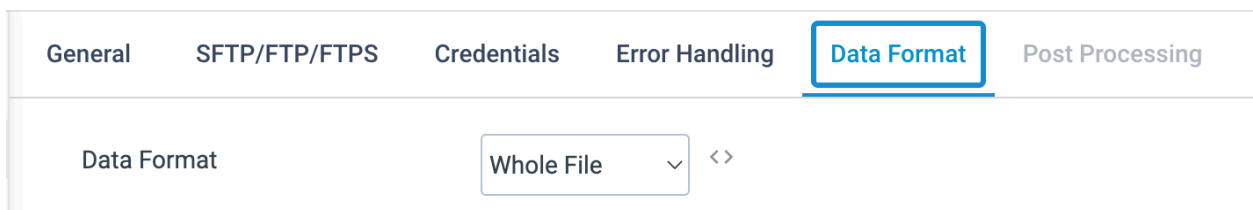
SFTP Origin Stage

The pipeline has an [SFTP Origin](#) configured like this:



General	SFTP/FTP/FTPS	Credentials	Error Handling	Data Format
Connection		None		
Resource URL ⓘ		<code>\${SFTP_BASE_URL}</code>		
Protocol ⓘ		SFTP ▾ <>		
Process Subdirectories ⓘ		<input type="checkbox"/>		
File Name Pattern Mode ⓘ		Glob ▾ <>		
File Name Pattern ⓘ		<code>\${SFTP_FILE_PATTERN}</code>		

The Data Format is pre-set to Whole File Data Format:



General	SFTP/FTP/FTPS	Credentials	Error Handling	Data Format	Post Processing
Data Format		Whole File ▾ <>			

S3 Staging Stage

The S3 Staging stage is an [S3 Destination](#) configured as follows:

S3 Staging

Show Advanced Options

Edit Mode

All Changes Saved

Realtime Summary

Configuration

Info

Schema

External Libraries

Help

General

Amazon S3

SSE

Advanced

Data Format

Connection

None

Authentication Method

Instance Profile <>

Assume Role ⓘ

☐

Use Specific Region ⓘ

☐

Bucket ⓘ

`${S3_STAGE_BUCKET}`

Common Prefix

`${S3_STAGE_BASE_DIR}`

Partition Prefix ⓘ

Partition Prefix

Object Name Suffix ⓘ

Object Name Suffix

Hide Advanced Options ^

Object Ownership ⓘ

Default <>

Delimiter

/

The Data Format is pre-set to Whole File Data Format and will preserve the source file names:

The screenshot shows the 'S3 Staging' configuration interface. At the top, there's a header with 'S3 Staging', a 'Show Advanced Options' toggle, an 'Edit Mode' button, and a status 'All Changes Saved'. Below the header, there's a sidebar on the left with navigation links: 'Realtime Summary', 'Configuration' (highlighted), 'Info', 'Schema', and 'External Libraries'. The main content area has tabs for 'General', 'Amazon S3', 'SSE', 'Advanced', and 'Data Format' (highlighted with a red box). In the 'Data Format' tab, there are two configuration fields, both highlighted with red boxes. The first field is 'Data Format' with an information icon and a dropdown menu set to 'Whole File'. The second field is 'File Name Expression' with an information icon and a text input containing the Jinja2 expression `${record:attribute('filename')}`.

If you want to ensure unique file names, you could append a timestamp or a UUID to the File Name Expression. Info on these and other functions is [here](#).

The S3 stage is configured to emit events, which will generate [Whole File Processed](#) events.

Get S3 File Name Stage

The `Get S3 File Name` stage is a [Jython Evaluator](#) configured to extract the S3 file name from the S3 Object Key. This step is needed in my example because the object key includes the base directory path of my Snowflake stage and I need to extract just the file name.

The current version of this example assumes files are written to the top level of the staging directory, though the example could easily be extended to recurse subdirectories. And there are many alternative implementations of this step that could be considered.

Here is the Jython script used in the example:

```
# Get the file name from the S3 objectKey.

for record in sdc.records:
    try:

        # Get the S3 objectKey
        object_key = record.value['targetFileInfo']['objectKey']

        # Split the objectKey path
        splits = object_key.split('/')

        # Grab the last split
        s3_file_name = splits[len(splits) - 1]

        # Assign the file_name to the record
        record.value['s3_file_name'] = s3_file_name

        # Write the record
        sdc.output.write(record)

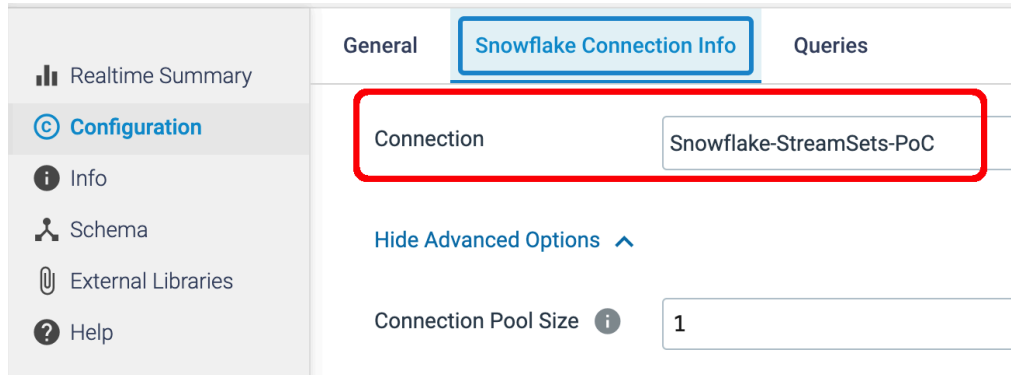
    except Exception as e:
        sdc.error.write(record, str(e))
```

The result is this stage appends a field named `s3_file_name` to each record. That field is used in the `COPY INTO` command performed in the next step

Copy File into Snowflake

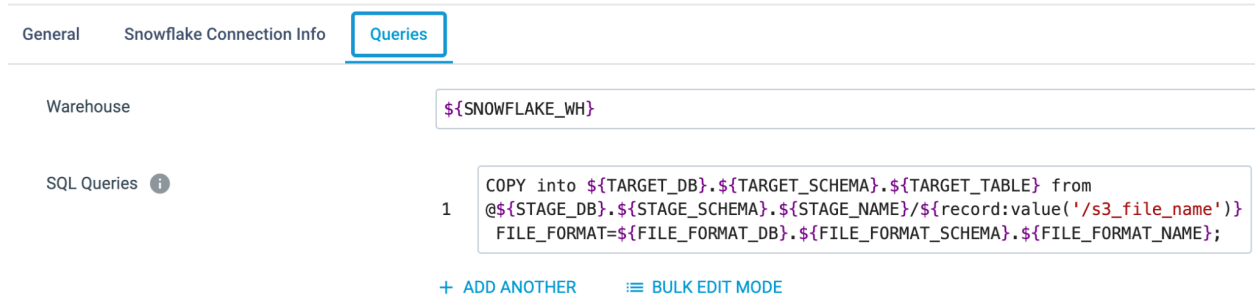
The Copy File into Snowflake stage is a [Snowflake Executor](#) configured like this:

I used a pre-defines Snowflake Connection:



The screenshot shows the 'Snowflake Connection Info' tab in the configuration interface. The 'Connection' field is highlighted with a red box and contains the value 'Snowflake-StreamSets-PoC'. The 'Connection Pool Size' is set to 1. The left sidebar shows the 'Configuration' tab selected.

And here is the heavily parameterized COPY INTO query:



The screenshot shows the 'Queries' tab in the configuration interface. The 'Warehouse' field is set to `${SNOWFLAKE_WH}`. The 'SQL Queries' field contains a parameterized COPY INTO query. Below the query, there are links for '+ ADD ANOTHER' and 'BULK EDIT MODE'.

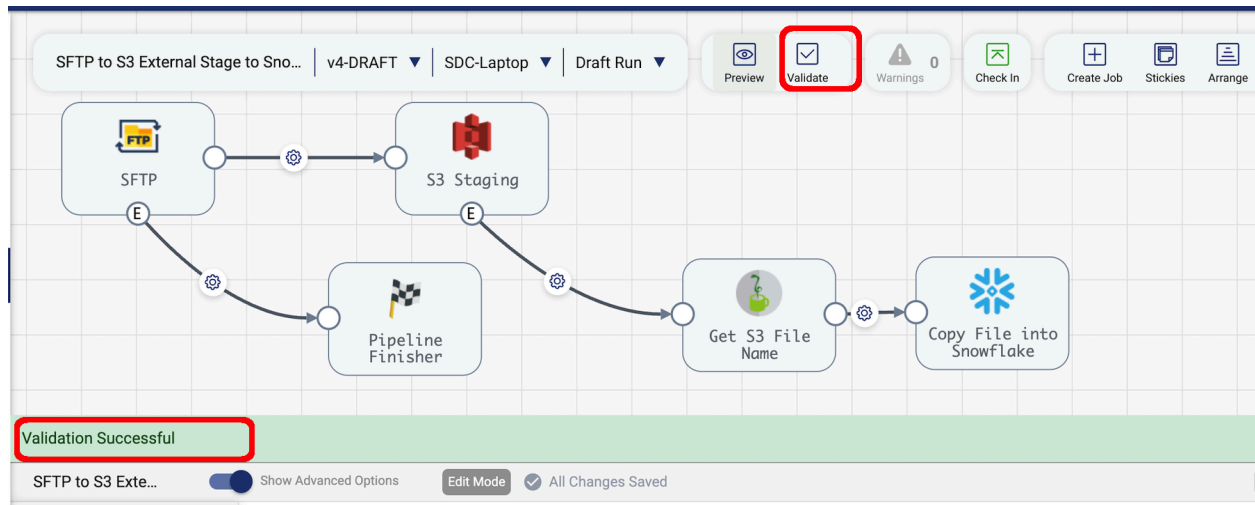
The SQL Query is this:

```
COPY into ${TARGET_DB}.${TARGET_SCHEMA}.${TARGET_TABLE} from
@${STAGE_DB}.${STAGE_SCHEMA}.${STAGE_NAME}/${record:value('/s3_file_name')}
FILE_FORMAT=${FILE_FORMAT_DB}.${FILE_FORMAT_SCHEMA}.${FILE_FORMAT_NAME};
```

Note that almost all values are parameterized in the query, and the name of the S3 file to COPY is retrieved from the `s3_file_name` field set by the previous stage.

Validate the Pipeline

Validate the pipeline and make sure the validation is successful:



Preview the Pipeline

Choose `Preview > Configure Preview` and unset the `Write to Destinations and Executors` checkbox, then click `Run Preview`

Preview Configuration

Preview Source:	<input type="text" value="Configured Source"/>
Preview Batch Size:	<input type="text" value="10"/>
Preview Timeout (in milliseconds):	<input type="text" value="120000"/>
Run Preview Through Stage:	<input type="text" value="S3 Staging"/>
Time zone:	<input type="text" value="Browser"/>
Write to Destinations and Executors:	<input type="checkbox"/>
Execute Pipeline Lifecycle Events:	<input checked="" type="checkbox"/>
Show Record/Field Header:	<input checked="" type="checkbox"/>
Show Field Type:	<input checked="" type="checkbox"/>
Save Preview Record Schema:	<input checked="" type="checkbox"/>

You should be able to confirm that the SFTP connector is reading at least one file. For example, in my environment I can see I'm able to pick up the file 1.csv:

The screenshot displays a data pipeline editor with a canvas showing the following stages: SFTP, S3 Staging, Pipeline Finisher, Get S3 File Name, and Copy File into Snowflake. The SFTP stage is selected, and the 'Preview Stage: SFTP' window is open. The 'OUTPUT' tab shows a single record with the following details:

```
Record1: {MAP}
  fileRef: {FILE_REF} [object Object]
  fileInfo: {MAP}
    file: {STRING} "/files/1.csv"
    filename: {STRING} "1.csv"
    size: {LONG} 17751839
    remoteUri: {STRING} "sftp://35.185.193.227:22/files"
    contentEncoding: {STRING}
    mtime: {LONG} 1707434618000
    contentType: {STRING}
```

The 'file' and 'filename' fields are highlighted with a red box. The interface also includes buttons for 'View Logs' and 'Close Preview' at the top, and a 'Single' button in the bottom right corner of the preview window.

Exit the preview.

Run the Pipeline

In my test, I have a single file named 1.csv on my SFTP server with 20,000 records in it.

Before I run the pipeline my S3 staging directory is empty, and my Snowflake target table has no rows in it.

Start the pipeline and when it has finished reading files from the SFTP server it will stop. In my environment, when the pipeline completes, I see the file in my S3 staging directory and 20,000 rows in my Snowflake table.

Click on the Snowflake stage to see it received one event and executed one COPY INTO command against Snowflake, and there were no errors:

