

Operationalizing a SQL Blob on Snowflake

| | |
|--|-----------|
| Introduction | 2 |
| Example Pipelines | 2 |
| Download Links | 3 |
| Import the Pipelines into your environment | 3 |
| Pipeline #1: Snowflake SQL Blob from File | 4 |
| Pipeline Parameters | 4 |
| SQL Files Stage | 5 |
| SQL Statement Parser Stage | 6 |
| Field Pivoter Stage | 6 |
| Snowflake Executor Stage | 7 |
| Set an Error Record Location | 8 |
| Example SQL File | 8 |
| Preview the Pipeline | 9 |
| Run the Pipeline | 10 |
| Pipeline #2: Snowflake SQL Blob from Pipeline | 11 |
| Operationalizing a Pipeline | 12 |
| Create a Job Template for the Pipeline | 12 |
| Create and Start a Job Template Instance | 16 |
| View Job History | 20 |
| Schedule the Job Template Instance | 20 |

Introduction

This document describes how to operationalize the execution of a SQL blob on Snowflake using StreamSets Platform. This might be considered an anti-pattern, as SQL blobs should be refactored to use Transformer for Snowflake's native stages, for intelligibility and ease of maintenance. However, the motivation for this example is the scenario where a Data Services team must respond quickly to their customer (for example a Data Science team) who throws SQL blobs over the fence and requires immediate operationalization of the SQL, with no time for refactoring, just "get this into production!"

All artifacts are available in this GitHub repo:

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

Example Pipelines

This example contains two pipelines:

- `Snowflake SQL Blob from File`

This pipeline retrieves SQL files from a directory reachable from the Data Collector machine, for example, a local directory or an NFS mounted directory.

The pipeline could easily be modified to pick up SQL files from an S3 bucket, SFTP, or other source system.

This approach is the recommended way to proceed, as a single pipeline can be re-used by multiple [Job Template Instances](#), with each instance pointing to a different SQL file, and with the SQL decoupled from the pipeline definition.

- `Snowflake SQL Blob from Pipeline.`

This pipeline has the SQL text embedded within the pipeline itself, which requires a separate pipeline per SQL blob.

This approach is, in general, not recommended, because the pipeline would have to be edited if the SQL changes, and handling multiple SQL blobs results in multiple copies of the pipeline, but has the advantage of having no external dependencies.

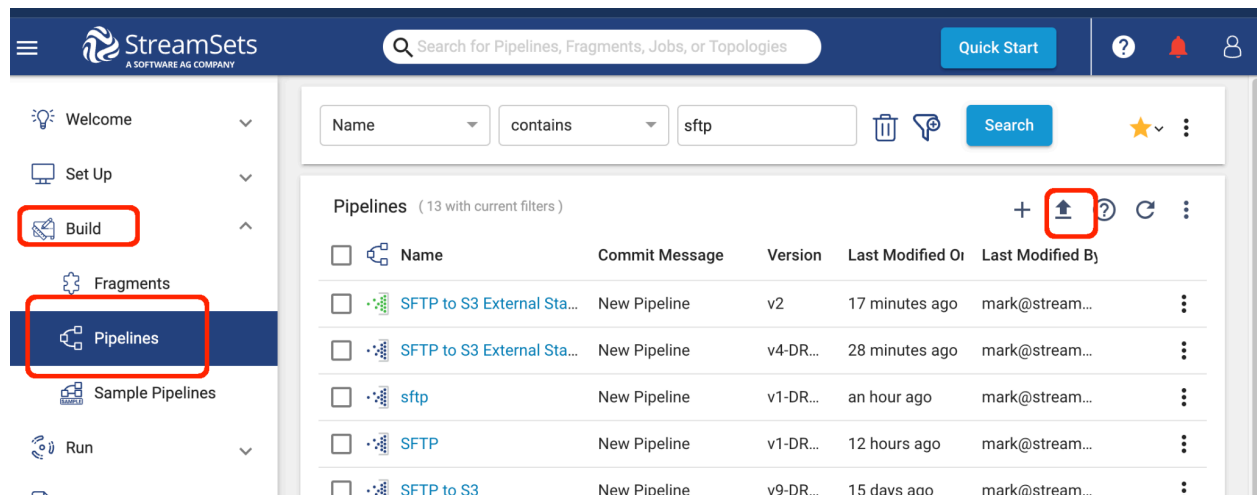
Download Links

The example pipeline `Snowflake SQL Blob from File` can be downloaded using the download button on [this page](#).

The example pipeline `Snowflake SQL Blob from Pipeline` can be downloaded using the download button on [this page](#).

Import the Pipelines into your environment

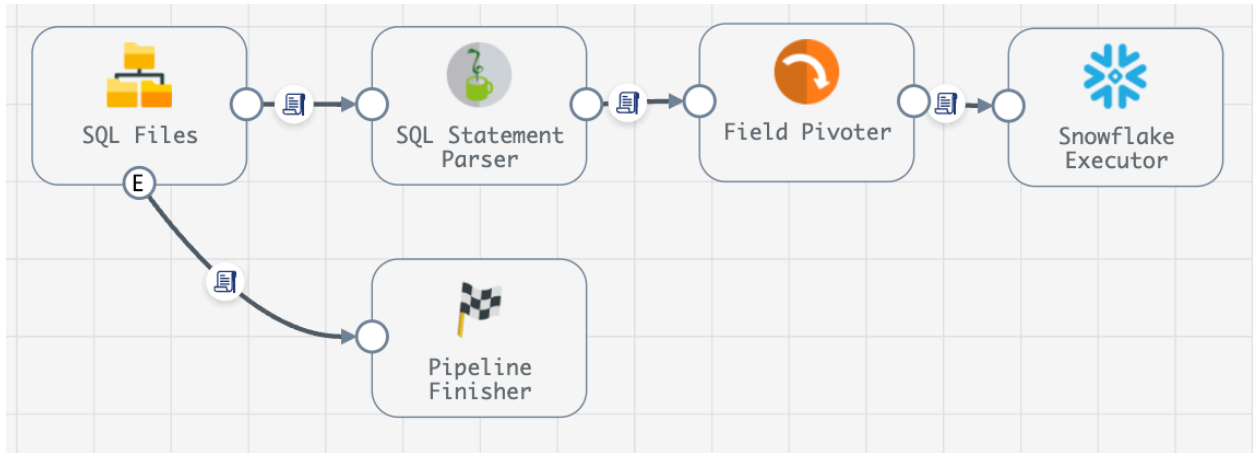
Use the upload button in the Pipeline List View to import the pipeline:



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

Pipeline #1: Snowflake SQL Blob from File

The pipeline looks like this:



Pipeline Parameters

The pipeline has the following parameters. Set the `SNOWFLAKE_WH` parameter to your warehouse, and set the `SQL_DIR` and `SQL_FILE` parameters to the local file system or NFS mount location where the SQL blob file is stored:

| General | Parameters | Notifications | Error Records | Advanced | Error Records | Test Origin |
|---|------------|---------------|---------------|------------------|---------------|-------------|
| Hide Advanced Options ^ | | | | | | |
| Parameters | | | | | | |
| | | SNOWFLAKE_WH | : | MARK_WH | | |
| | | SQL_DIR | : | /path/to/sql/dir | | |
| | | SQL_FILE | : | example.sql | | |

SQL Files Stage

The SQL Files stage is a [Directory Origin](#) configured like this::

General **Files** Post Processing Data Format

Files Directory ⓘ `${SQL_DIR}`

File Name Pattern ⓘ `${SQL_FILE}`

First File to Process ⓘ First File to Process

[Hide Advanced Options](#) ^

Number of Threads ⓘ 1

File Name Pattern Mode ⓘ Glob ▾ <>

Read Order ⓘ Lexicographically Ascending File Names ▾ <>

Batch Size (recs) ⓘ 1

The Data Format is set to Whole File:

General Files Post Processing **Data Format**

Data Format ⓘ Whole File ▾ <>

SQL Statement Parser Stage




The `SQL Statement Parser` stage is a [Jython Evaluator](#) that parses the SQL blob into a list of SQL statements. It assumes statements are delimited with semicolons and it ignores semicolons within single quoted string values.

See the Jython script within the pipeline for implementation details.

Field Pivoter Stage

A [Field Pivoter](#) stage is used to convert the list of SQL statements to a set of records with one SQL statement each, which are then routed in order to the Snowflake Executor.

Here is the configuration of the Field Pivoter that pivots on the `sql_commands` field and emits multiple `sql_command` records:

| General | Field Pivot |
|--|--|
| Field To Pivot  | <input type="text" value="/sql_commands"/> |
| Copy All Fields  | <input checked="" type="checkbox"/> |
| Pivoted Items Path  | <input type="text" value="/sql_command"/> |

Snowflake Executor Stage

Set your own connection in the Snowflake Connection Info tab:

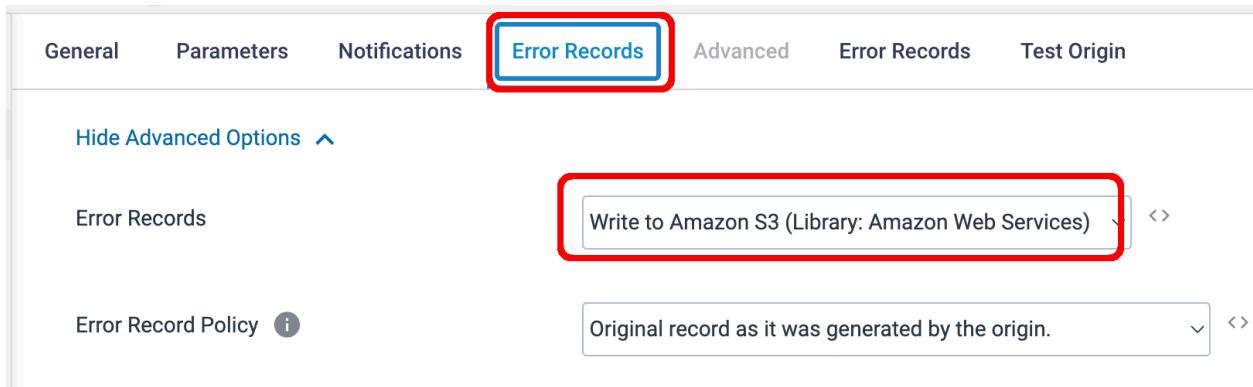
| | | |
|-------------------------|---------------------------|--------------------------|
| General | Snowflake Connection Info | Queries |
| Connection | | Snowflake-StreamSets-PoC |
| Hide Advanced Options ^ | | |
| Connection Pool Size ⓘ | | 1 |

The stage is configured to execute each record's `sql_command`:

| | | |
|---------------|---------------------------|---|
| General | Snowflake Connection Info | Queries |
| Warehouse | | <code>\${SNOWFLAKE_WH}</code> |
| SQL Queries ⓘ | | 1 <code>\${record:value('/sql_command')}</code> |
| | | + ADD ANOTHER ≡ BULK EDIT MODE |

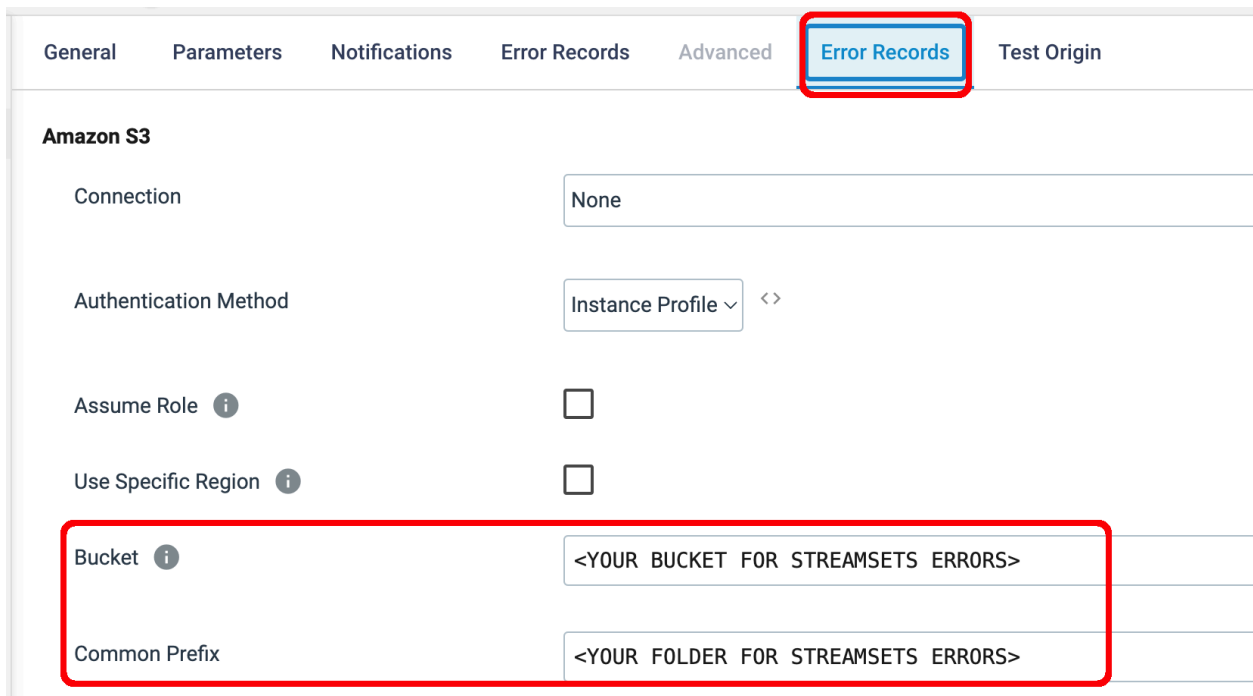
Set an Error Record Location

Set the error record location to S3:



The screenshot shows the 'Error Records' tab selected in a configuration interface. The 'Error Records' dropdown menu is open, showing the option 'Write to Amazon S3 (Library: Amazon Web Services)' selected. The 'Error Record Policy' dropdown menu is also open, showing the option '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 'Amazon S3' section is expanded, showing the following settings:

- Connection: None
- Authentication Method: Instance Profile
- Assume Role: ☐
- Use Specific Region: ☐
- Bucket: <YOUR BUCKET FOR STREAMSETS ERRORS>
- Common Prefix: <YOUR FOLDER FOR STREAMSETS ERRORS>

Example SQL File

I used a SQL blob file with [this content](#) for my tests.

Preview the Pipeline

In preview mode, with the Field Pivoter stage selected we can see the SQL blob has been parsed into three SQL statements:

Snowflake SQL Blob from File (146)

Close Preview

SQL Files

SQL Statement Parser

Field Pivoter

Snowflake Executor

Preview Stage: Field Pivoter

Single Multiple

Output Order Match with Input Order

INPUT

Record1: {MAP}

fileRef: {FILE_REF} [object Object]

fileInfo: {MAP}

sql_commands: {LIST [3]}

- 0: {STRING} "create table MARK_DB.MARK_SCHEMA.T_1(C_1 STRING);"
- 1: {STRING} "insert into MARK_DB.MARK_SCHEMA.T_1 values ('a');"
- 2: {STRING} "update MARK_DB.MARK_SCHEMA.T_1 set C_1 = 'b' where C_1 = 'a';"

Record Header

OUTPUT

Record1-Output Record1: {MAP}

fileRef: {FILE_REF} [object Object]

fileInfo: {MAP}

sql_command: {STRING} "create table MARK_DB.MARK_SCHEMA.T_1(C_1 STRING);"

Record Header

Record1-Output Record2: {MAP}

fileRef: {FILE_REF} [object Object]

fileInfo: {MAP}

sql_command: {STRING} "insert into MARK_DB.MARK_SCHEMA.T_1 values ('a');"

Record Header

Record1-Output Record3: {MAP}

fileRef: {FILE_REF} [object Object]

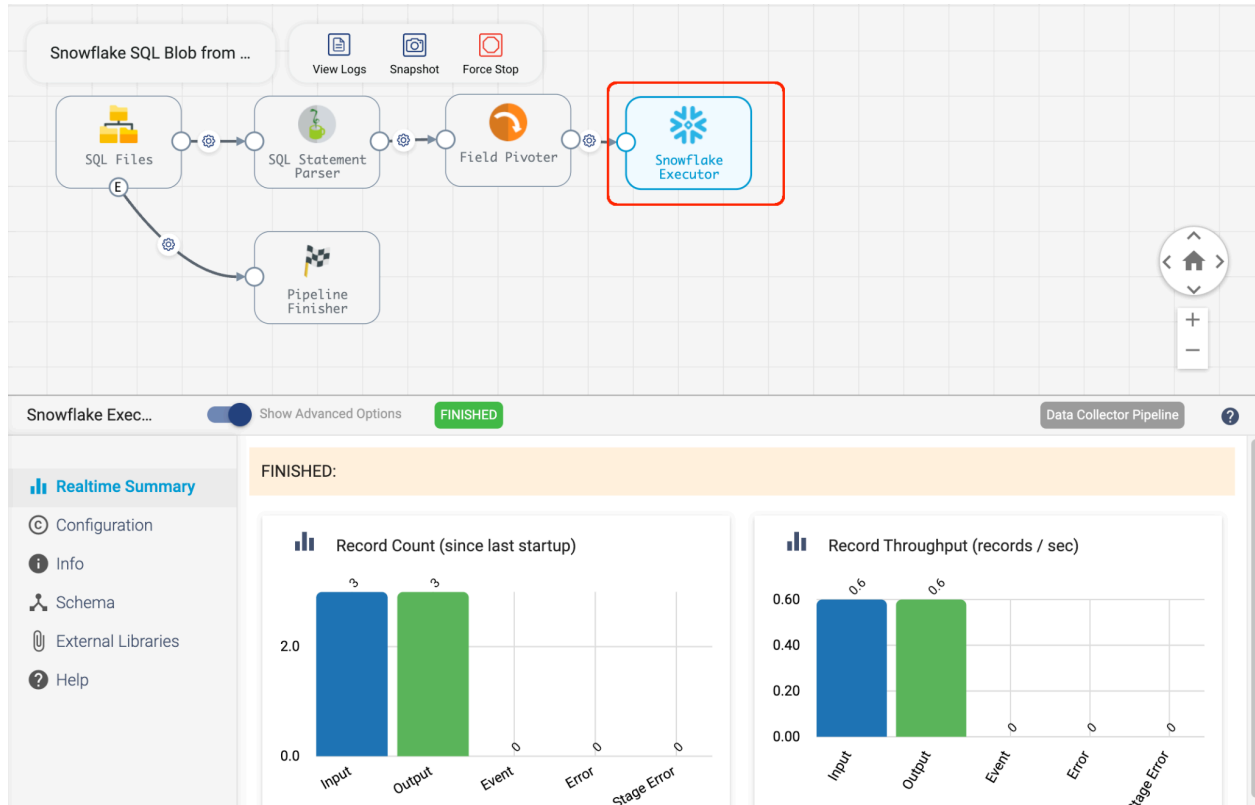
fileInfo: {MAP}

sql_command: {STRING} "update MARK_DB.MARK_SCHEMA.T_1 set C_1 = 'b' where C_1 = 'a';"

Record Header

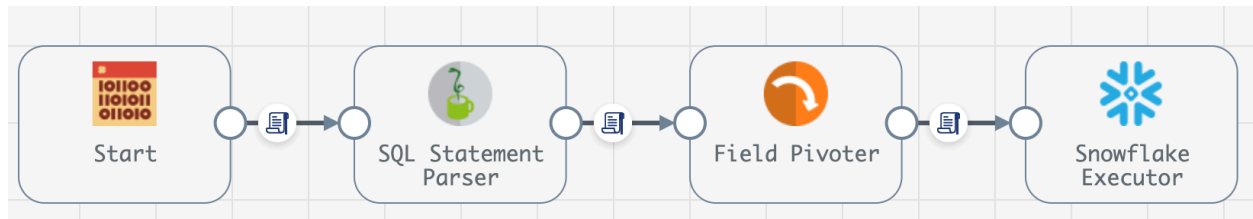
Run the Pipeline

Run the pipeline and you should see three SQL command records successfully handled by the Snowflake Executor. The pipeline will automatically stop when it is complete.



Pipeline #2: Snowflake SQL Blob from Pipeline

The pipeline looks like this:



This pipeline is identical to the first pipeline, with the exception of having a trivial Dev Raw Data origin configured like this (and the event it emits is ignored, we just need a record to kick the processing off):

| General | Raw Data | Event Data | Data Format |
|------------------------|---|------------|-------------|
| Raw Data | <div><div>1 {</div><div>2 "event": "start"</div><div>3 }</div></div> <p>Press F11 or Cmd+B when cursor is in the editor to toggle full screen</p> | | |
| Stop After First Batch | <input checked="" type="checkbox"/> | | |

The SQL blob is pasted into the top of the Jython script of the SQL Statement Parser stage as a multi-line String like this:

| Script |
|---|
| <pre>1 # The SQL Blob as a multiline string 2 sql = """ 3 create table 4 "MARK_DB"."MARK_SCHEMA"."T_1" (5 C_1 STRING 6); 7 8 insert into "MARK_DB"."MARK_SCHEMA"."T_1" 9 values ('a'); 10 11 update "MARK_DB"."MARK_SCHEMA"."T_1" 12 set C_1 = 'b' 13 where C_1 = 'a'; 14 """</pre> |

See the Jython stage within the pipeline for the full script.

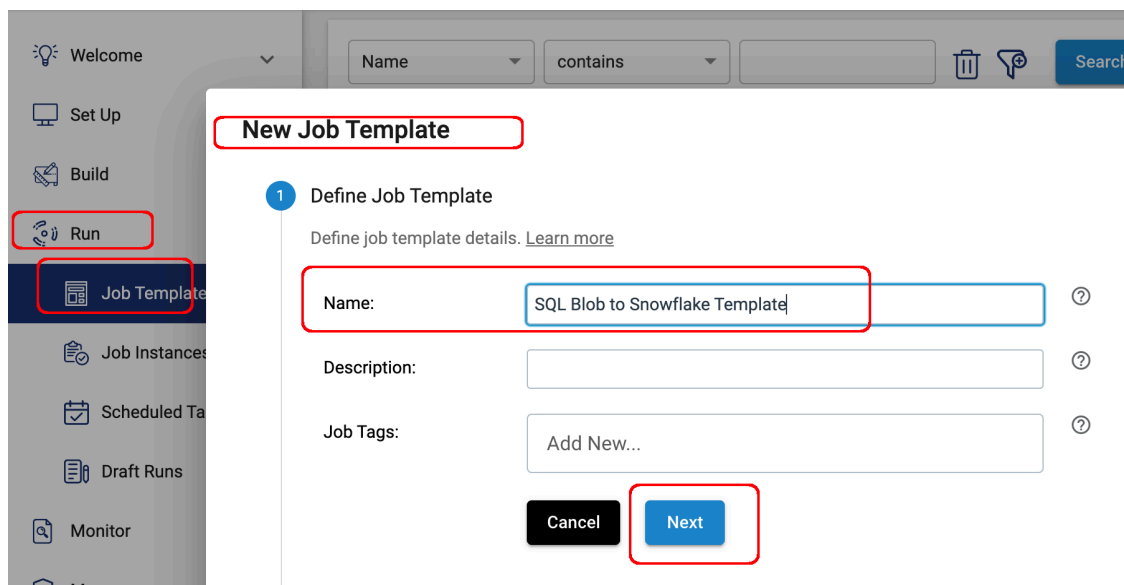
This pipeline can be previewed and run just like the previous pipeline.

Operationalizing a Pipeline

I'll use the first pipeline (Snowflake SQL Blob from File) in the examples below. To operationalize the pipeline we'll first create a Job Template for it, then create and launch an instance of the Job Template configured to process a particular SQL blob file (1.sql), and finally, we'll schedule the Job Template Instance to run every weekday.

Create a Job Template for the Pipeline

Create a new Job Template named `SQL Blob to Snowflake Template`:



The screenshot shows the 'New Job Template' dialog box in a Snowflake web interface. The left sidebar contains navigation items: 'Welcome', 'Set Up', 'Build', 'Run' (highlighted with a red box), 'Job Template' (highlighted with a red box), 'Job Instances', 'Scheduled Tasks', 'Draft Runs', and 'Monitor'. The main dialog has a title bar 'New Job Template' and a step indicator '1 Define Job Template'. Below the title bar, it says 'Define job template details. [Learn more](#)'. The form contains three fields: 'Name' (with the value 'SQL Blob to Snowflake Template' and a red box around it), 'Description' (empty), and 'Job Tags' (with a placeholder 'Add New...'). At the bottom, there are 'Cancel' and 'Next' buttons, with the 'Next' button highlighted by a red box.

Click Next.

Pick the latest version of the Snowflake SQL Blob from File pipeline

New Job Template

1 Define Job Template

2 Select Pipeline

Select the published pipeline that you want to run. [Learn more](#)

Pipeline: Snowflake SQL Blob from File

[Click here to select](#)

Pipeline Version: v8 [Click here to select](#)

Back

Next

Click Next.

Pick the deployment and engine label Job Template Instances should run on, and enable failover

New Job Template

3 Configure Job Details

Configure job details to determine how engines run the pipeline. The default values for the advanced options should work in most cases. [Learn more](#)

Deployment:

SDC-Laptop (Self-Managed)

Engine Labels:

sdclaptop

Add New...

Enable Failover:

☒

[Show Advanced Options](#) ▼

Back

Save & Next

Save & Exit

Click Save and Next

Clear the parameter values. This is so the Job Template itself does not have a default SQL blob file associated with it; those values will be set when we create instances of the Job Template in a subsequent step.

New Job Template

3 Configure Job Details

4 Set Parameter Defaults

Define the parameter values to start the pipeline with. Override the default values using simple or bulk edit mode. In bulk edit mode, configure parameter values in JSON format. [Learn more](#)

| Parameter Name | Default Value | Static Parameter |
|----------------|--|--------------------------|
| SQL_DIR : | <input type="text" value="Enter Value"/> | <input type="checkbox"/> |
| SQL_FILE : | <input type="text" value="Enter Value"/> | <input type="checkbox"/> |

 BULK EDIT MODE

Back

Save & Next

Save & Exit

Click Save and Next and then Exit.

You should see the Job Template that was just created in the Job Template list:

Welcome

Set Up



Build

Run

Job Templates

Job Instances

Name contains

 Search

Job Templates (18)

| <input type="checkbox"/> | Name | Pipeline |
|--------------------------|--------------------------------|------------------------------|
| <input type="checkbox"/> | SQL Blob to Snowflake Template | Snowflake SQL Blob from File |
| <input type="checkbox"/> | Oracle CDC to Snowflake | Oracle CDC to Snowflake |
| <input type="checkbox"/> | REST API | REST API to Google Cloud Sto |

Create and Start a Job Template Instance

To launch a Job to pick up and process a particular SQL blob file, create and start an instance of the Job Template parameterized for the given SQL blob file.

Click on the Job Template and then click the `Create and Start Job Instances` button:

[Job Templates](#) > SQL Blob to Snowflake Template

Job Template Details

 **Create & Start Job Instances**

Give the instance a name:

Create Job Instances

1 Define Job

Define job details. [Learn more](#)

Name:

SQL Blob #1 to Snowflake

Description:

Job Tags:

Add New...

Cancel

Next

Click Next

Keep these default settings:

Create Job Instances

1 Define Job

2 Select Job Template

Select the job template and optionally configure advanced options. [Learn more](#)

Job Template: SQL Blob to Snowflake Template [Click here to select](#) ?

Hide Advanced Options ^

Delete from Job Instances List ☐

when Completed:

Attach Instances to Template: ☒

Inherit Permissions: ☐

Back

Next

Click Next

Set the Instance Name Suffix to the Parameter Value named `SQL_FILE` so we will be able to tell which file is being processed by this Job Template Instance, and set the parameter values for the directory and filename of the SQL blob file you want to process:

Create Job Instances

3 Define Runtime Parameters

Define the parameter values to start the pipeline with. Override the default values using simple or bulk edit mode. In bulk edit mode, configure parameter values in JSON format. [Learn more](#)

Instance Name Suffix:

Parameter Value

?

Parameter Name:

SQL_FILE

?

Runtime Parameters for Each Instance:

Simple Edit

Bulk Edit

From File

1

Instance 1

SQL_DIR:

/Users/mark/mnt/sql

SQL_FILE:

1.sql

+

ADD ANOTHER INSTANCE

Back

Create & Start (1)

Click the Create and Start button.

You should see a message like this:

Create Job Instances

2 Select Job Template

3 Define Runtime Parameters

4 Review & Start

Successfully created & started 1 job instances.

Exit

If you quickly switch over to the Job Instances list you may catch the Job while it is still running:

The screenshot shows the 'Job Instances' page in a web application. On the left is a sidebar with navigation items: 'Welcome', 'Set Up', 'Build', 'Run', 'Job Templates', and 'Job Instances' (which is highlighted with a red box). The main area has a search bar at the top. Below it, a table lists job instances. The first instance, 'SQL Blob #1 to Snowflake - 1.sql', is selected (checkbox checked) and highlighted with a red box. Its 'Job Status' is 'ACTIVE' and 'Pipeline Status' is 'RUNNING'. Other instances below it are '[Prod] Job for dev to trash' and '[Dev] Job for dev to trash', both with 'INACTIVE' job status.

| Name | Pipeline | Version | Last Modified On | Job Status | Pipeline Status |
|--|----------------------------------|---------|------------------|------------|-----------------|
| <input checked="" type="checkbox"/> SQL Blob #1 to Snowflake - 1.sql | Snowflake SQL Blob from File ... | v8 | 2 minutes ago | ACTIVE | RUNNING |
| <input type="checkbox"/> [Prod] Job for dev to trash | dev to trash | v4 | 2 days ago | INACTIVE | |
| <input type="checkbox"/> [Dev] Job for dev to trash | dev to trash | v4 | 2 days ago | INACTIVE | |

Once the Job completes it will have either an `Inactive` status (if it completed successfully) , or an error status. For example:

This screenshot shows the 'Job Instances' page after the job has completed. The table now shows the 'SQL Blob #1 to Snowflake - 1.sql' job with an 'INACTIVE' status. The header indicates 'Job Instances (156 with current filters)'.

| Name | Pipeline | Version | Last Modified On | Job Status |
|---|----------------------------------|---------|------------------|------------|
| <input type="checkbox"/> SQL Blob #1 to Snowflake - 1.sql | Snowflake SQL Blob from File ... | v8 | 6 minutes ago | INACTIVE |

Note the Job has the suffix `1.sql` - the name of the SQL blob file it processed.

You can rerun this Job Template Instance by choosing the Start Job menu item:

This screenshot shows the 'Job Instances' page with the 'SQL Blob #1 to Snowflake - 1.sql' job selected. A red box highlights the job row. Another red box highlights the three-dot menu icon at the end of the row. A third red box highlights the 'Start Job' option in the dropdown menu that appears.

| Name | Pipeline | Version | Last Modified On | Job Status | Pipeline Status |
|--|----------------------------------|---------|------------------|------------|-----------------|
| <input checked="" type="checkbox"/> SQL Blob #1 to Snowflake - 1.sql | Snowflake SQL Blob from File ... | v8 | 10 minutes ago | INACTIVE | |
| <input type="checkbox"/> [Prod] Job for dev to trash | dev to trash | v4 | 2 days ago | INACTIVE | |
| <input type="checkbox"/> [Dev] Job for dev to trash | dev to trash | v4 | 2 days ago | INACTIVE | |

View Job History

Drill down on the Job Template Instance to see its run history:

| Job History | | | | | | | | | |
|-------------|----------|----------------|----------------|---------------|-------|--------|-------|---------------------------|--------|
| Run Count | Status | Start Time | Finish Time | Duration | Input | Output | Error | Summary | Commit |
| 5 | INACTIVE | Feb 10, 202... | Feb 10, 202... | a few seco... | 1 | 6 | 0 | View Summ | v8 |
| 4 | INACTIVE | Feb 10, 202... | Feb 10, 202... | a few seco... | 1 | 5 | 1 | View Summ | v8 |
| 3 | INACTIVE | Feb 10, 202... | Feb 10, 202... | a few seco... | 1 | 6 | 0 | View Summ | v8 |

Click the View Summary link for any run to see the details.

Schedule the Job Template Instance

Schedule the Job by clicking the Schedule Job link:

| Job Instances (156 with current filters) | | | | | | | + ↑ ? ↺ ⋮ |
|--|----------------------------------|----------------------------------|--------|--------------------|------------|-----------------|----------------|
| <input type="checkbox"/> | Name | Pipeline | Versio | Last Modified On ↓ | Job Status | Pipeline Status | |
| <input type="checkbox"/> | SQL Blob #1 to Snowflake - 1.sql | Snowflake SQL Blob from File ... | v8 | 14 minutes ago | INACTIVE | | ⋮ |
| <input type="checkbox"/> | [Prod] Job for dev to trash | dev to trash | v4 | 2 days ago | INACTIVE | | ▶ Start Job |
| <input type="checkbox"/> | [Dev] Job for dev to trash | dev to trash | v4 | 2 days ago | INACTIVE | | 🕒 Schedule Job |
| <input type="checkbox"/> | Job for Small Template Data | Small Template Data | v1 | 2 days ago | INACTIVE | | |

You will see this screen:

Select Job

Job Name

SQL Blob #1 to Snowflake - 1.sql

Job Name

3edec9ff-4d99-4acd-ac50-d4d126d50e4e:8030c2e9-1a39-11ec-a5fe-97c8d4369386

1 / 2

Cancel

Back

Next

Click the Next button

Set the start time(s) you want:

Action

START

Cron Expression

Minutes

Hourly

Daily

Weekly

Monthly

Yearly

Advanced

☒

Monday

☒

Tuesday

☒

Wednesday

☒

Thursday

☒

Friday

☐

Saturday

☐

Sunday

at 3

5


5 3 ? * MON,TUE,WED,THU,FRI *

Cancel

Back

Save

Click Save, and you should see a task has been created in the StreamSets Scheduler, with the Next Execution Time:

StreamSets
A SOFTWARE AS COMPANY

Platform

Control Hub

Search for Pipelines, Fragments, Jobs, or Topologies

Welcome

Set Up

Build

Run

Job Templates

Job Instances

Scheduled Tasks

Scheduled Tasks (3)

☐

Name ↑

Cron Expression

Time Zone

Status

Next Execution Time

☐ SQL Blob #1 to Snowflake - 1.sql

5 3 ? * MON,TUE,WED,THU,FF

UTC

Running

2/11/24, 7:05 PM

☐ Claims to Snowflake

0 0 1/1 * ? *

UTC

Paused

2/11/24, 4:00 PM

☐ ADLS Data Prep

0 0 1/1 * ? *

UTC

Paused

2/11/24, 4:00 PM

Items per page: 501 - 3 of 3