Operationalizing a SQL Blob on Snowflake

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.

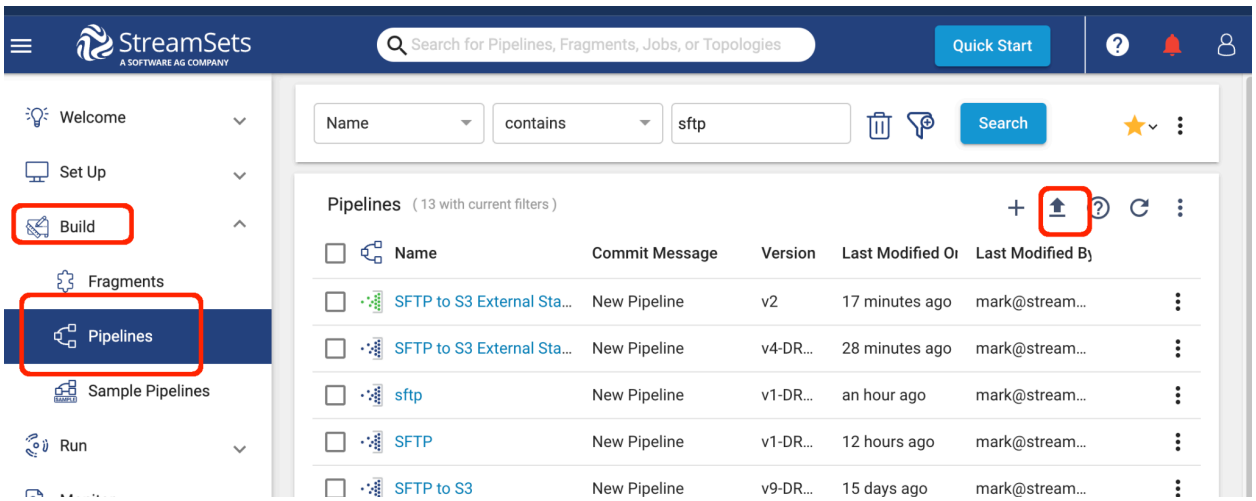
Download Links

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

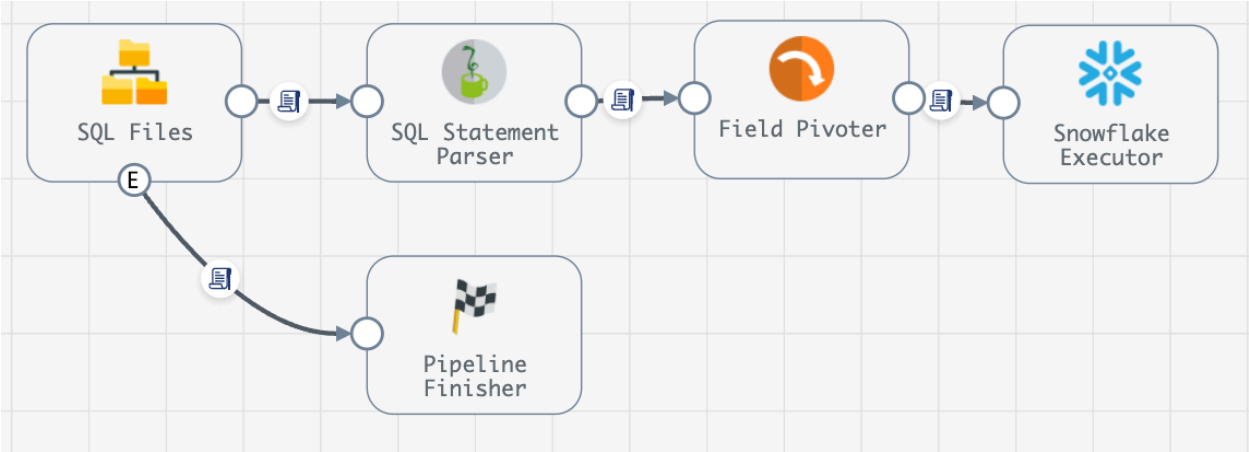
Use the upload button in the Pipeline List View to import the pipeline:



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

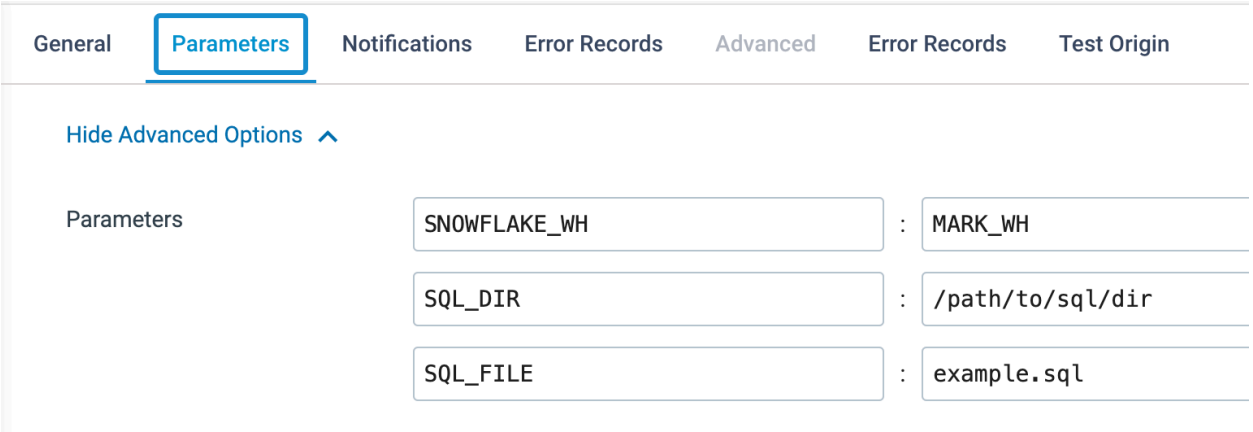
Pipeline #1: Snowflake SQL Blob from File

The pipeline looks like this:



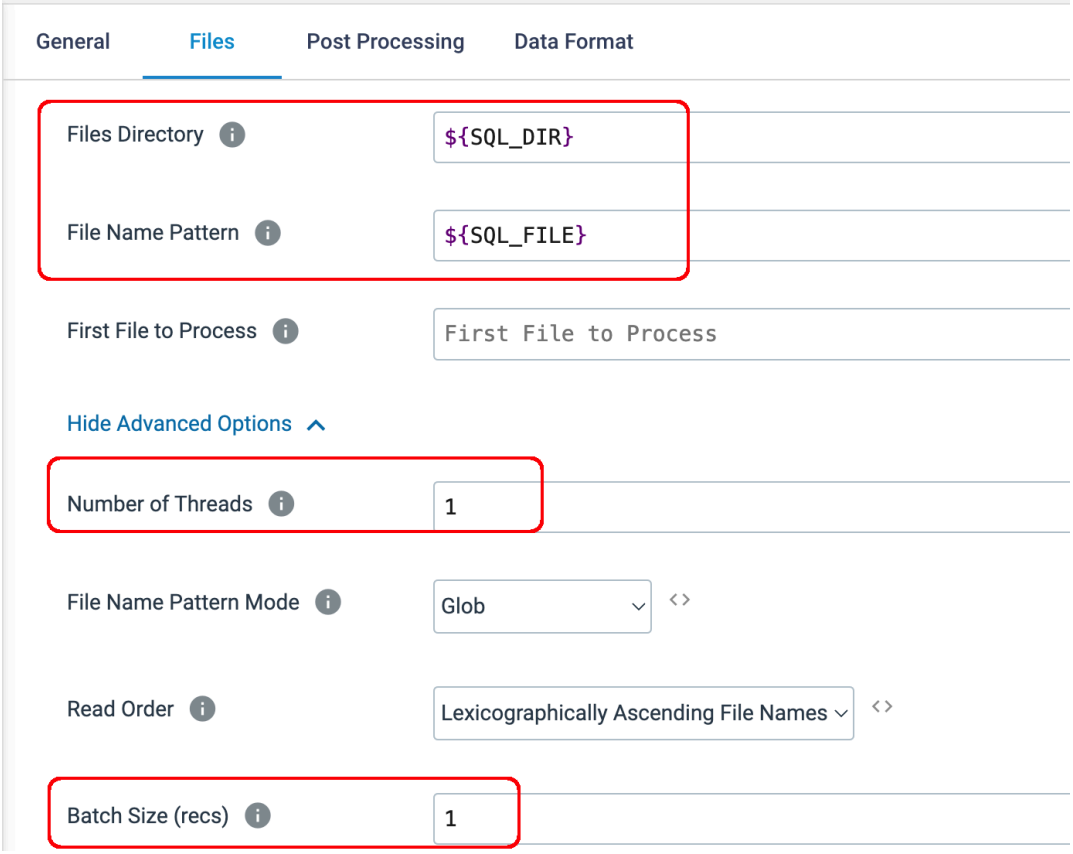
Pipeline Parameters

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:

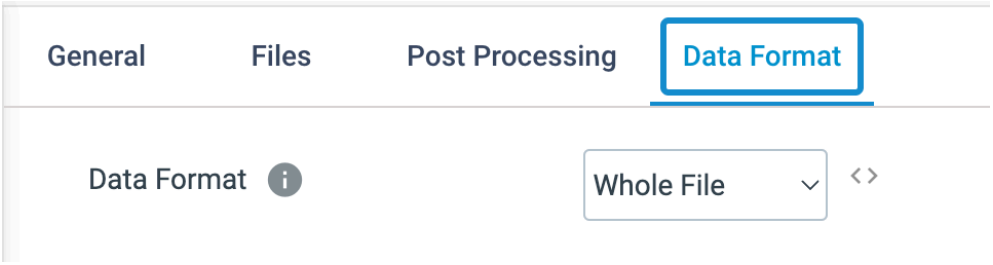


SQL Files Stage

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



The Data Format is set to Whole File:



SQL Statement Parser Stage

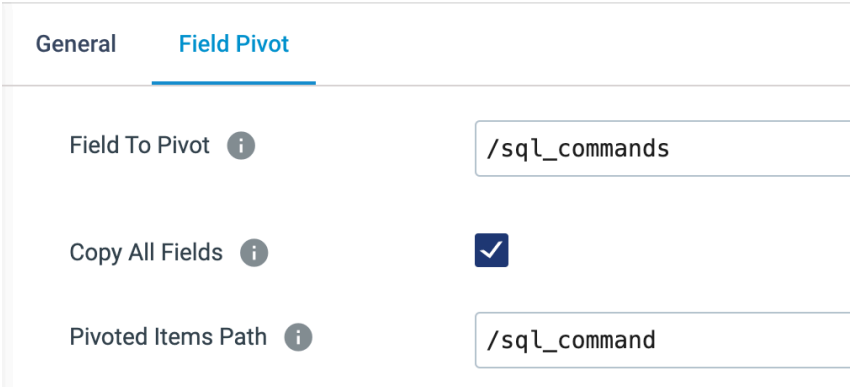
The `SQL Statement Parser` stage is a Jython Evaluator that parses the SQL blob into a list of SQL statements. It assumes statements are delimited with semicolons and it ignores semicolons within single quoted string values.

See the Jython script within the pipeline for implementation details.

Field Pivoter Stage

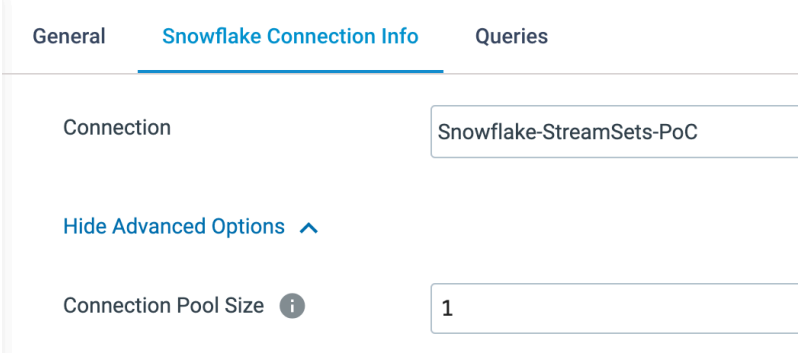
A Field Pivoter stage is used to convert the list of SQL statements to a set of records with one SQL statement each, which are then routed in order to the Snowflake Executor.

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

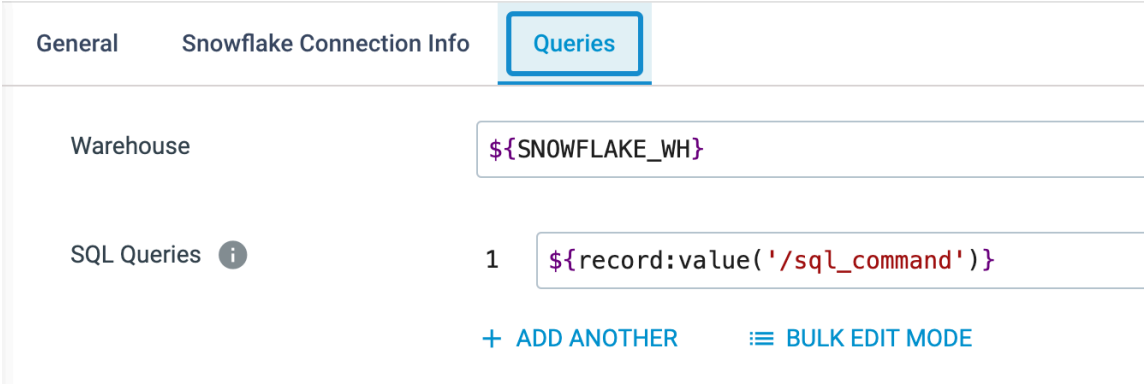


Snowflake Executor Stage

Set your own connection in the Snowflake Connection Info tab:

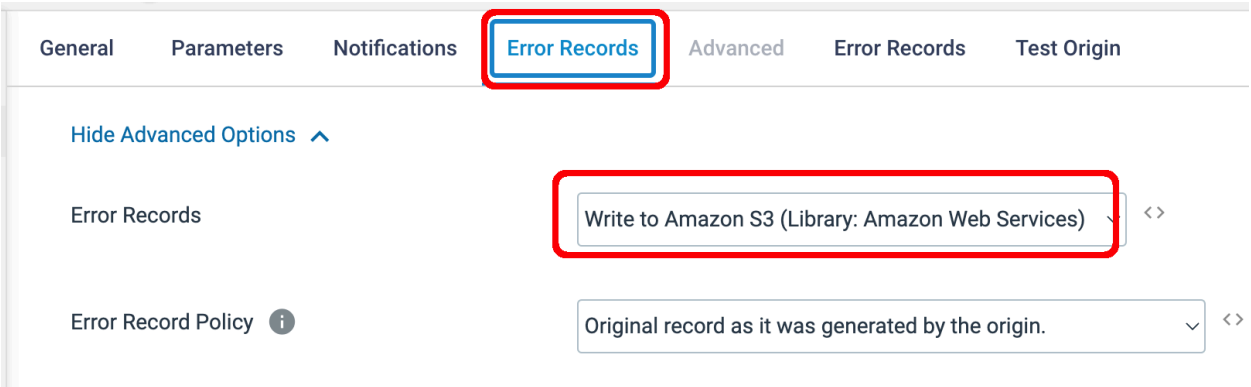


The stage is configured to execute each record's `sql_command`:

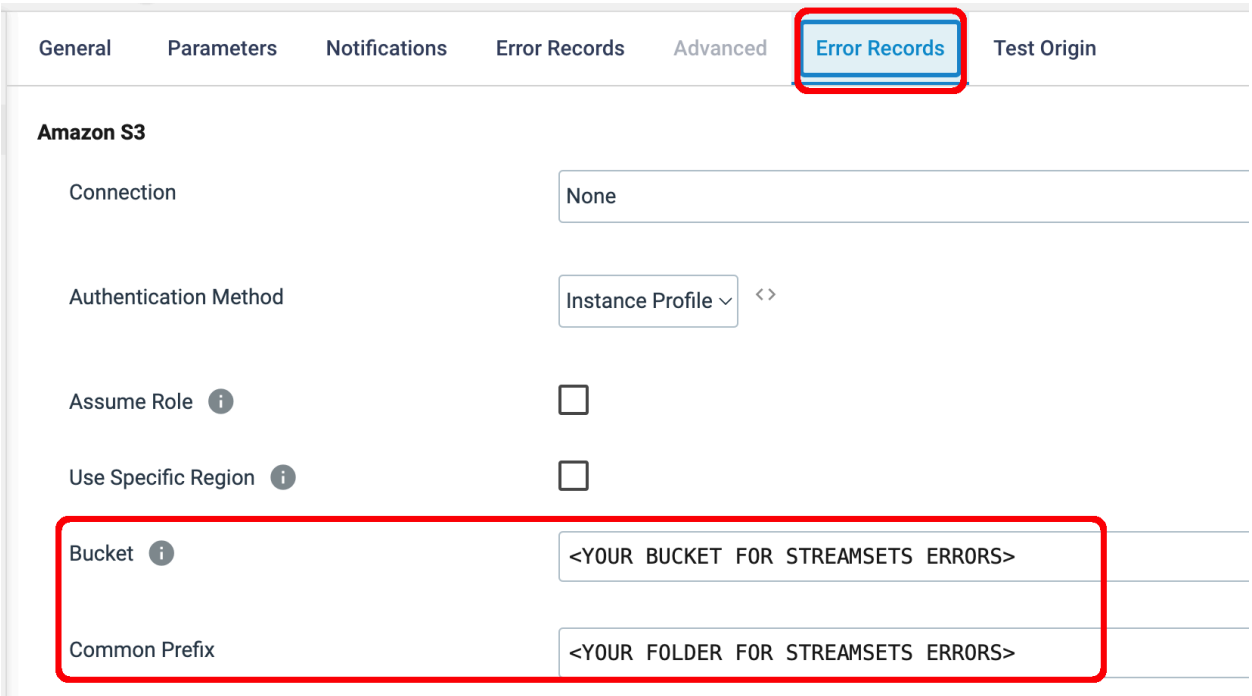


Set an Error Record Location

Set the error record location to S3:



Set an S3 Bucket and folder to hold error records:

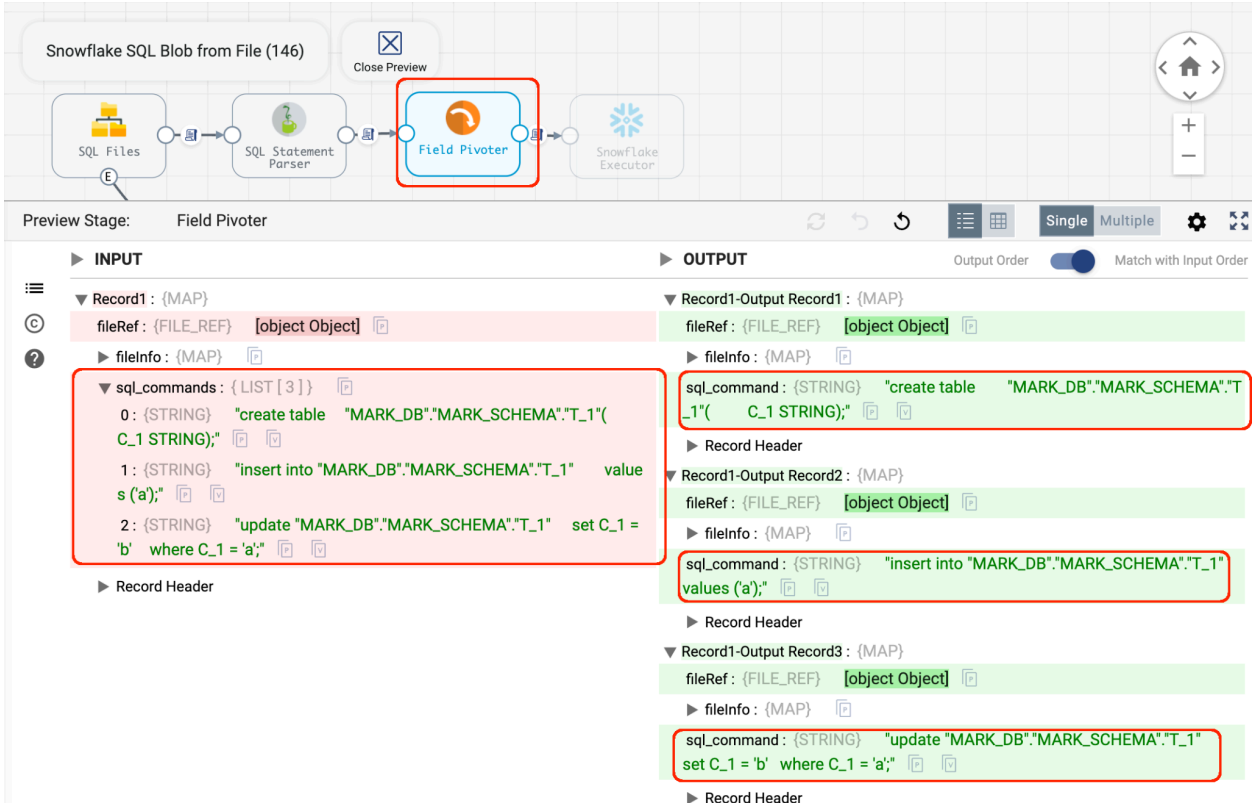


Example SQL File

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

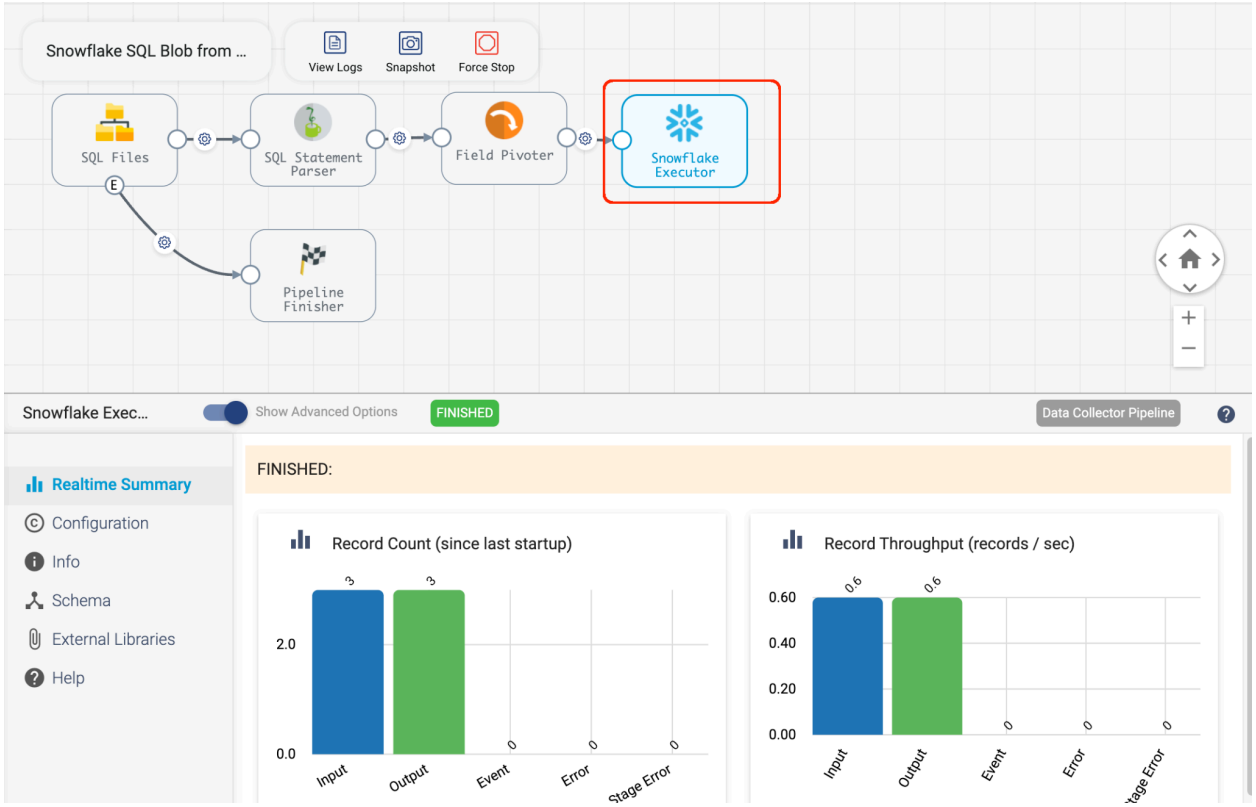
Preview the Pipeline

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



Run the Pipeline

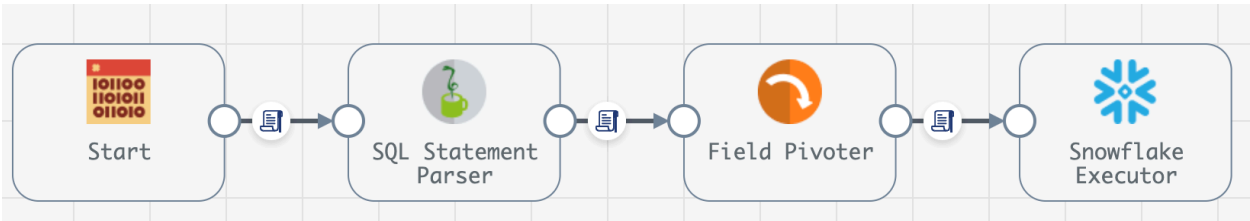
Run the pipeline and you should see three SQL command records successfully handled by the Snowflake Executor. The pipeline will automatically stop when it is complete.



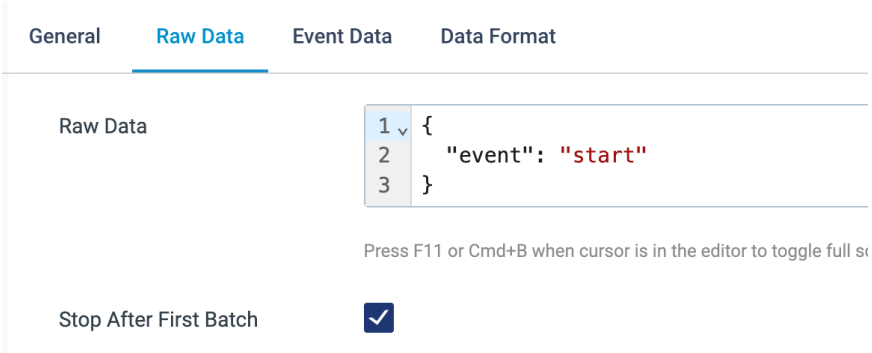
:

Pipeline #2: Snowflake SQL Blob from Pipeline

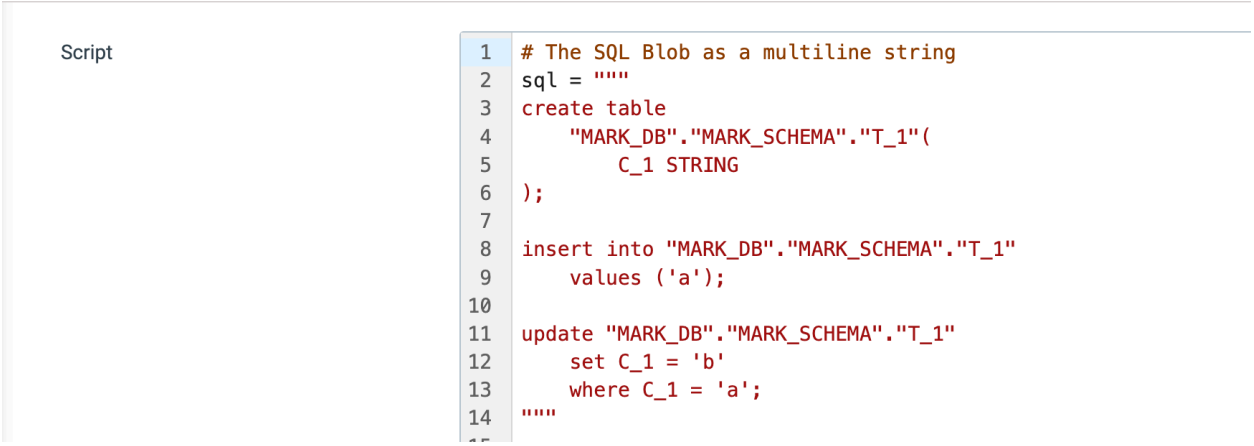
The pipeline looks like this:



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):



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:



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

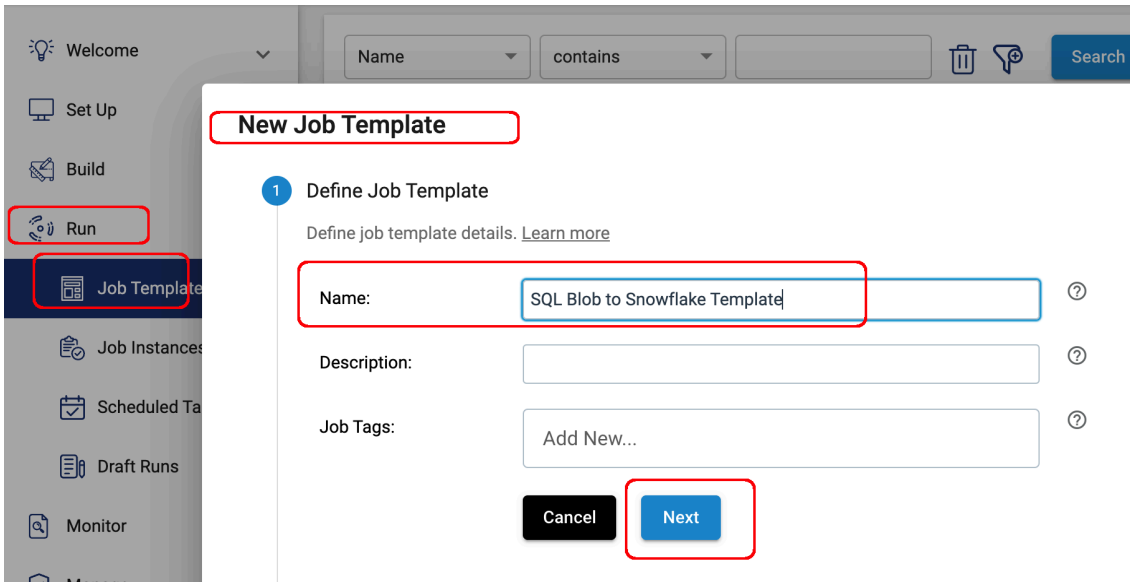
This pipeline can be previewed and run just like the previous pipeline.

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.

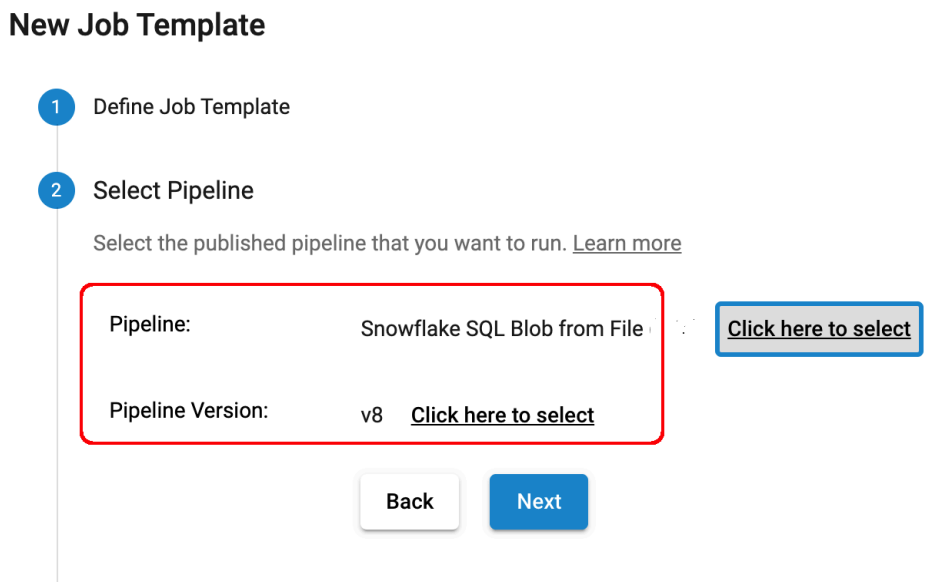
Create a Job Template for the Pipeline

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



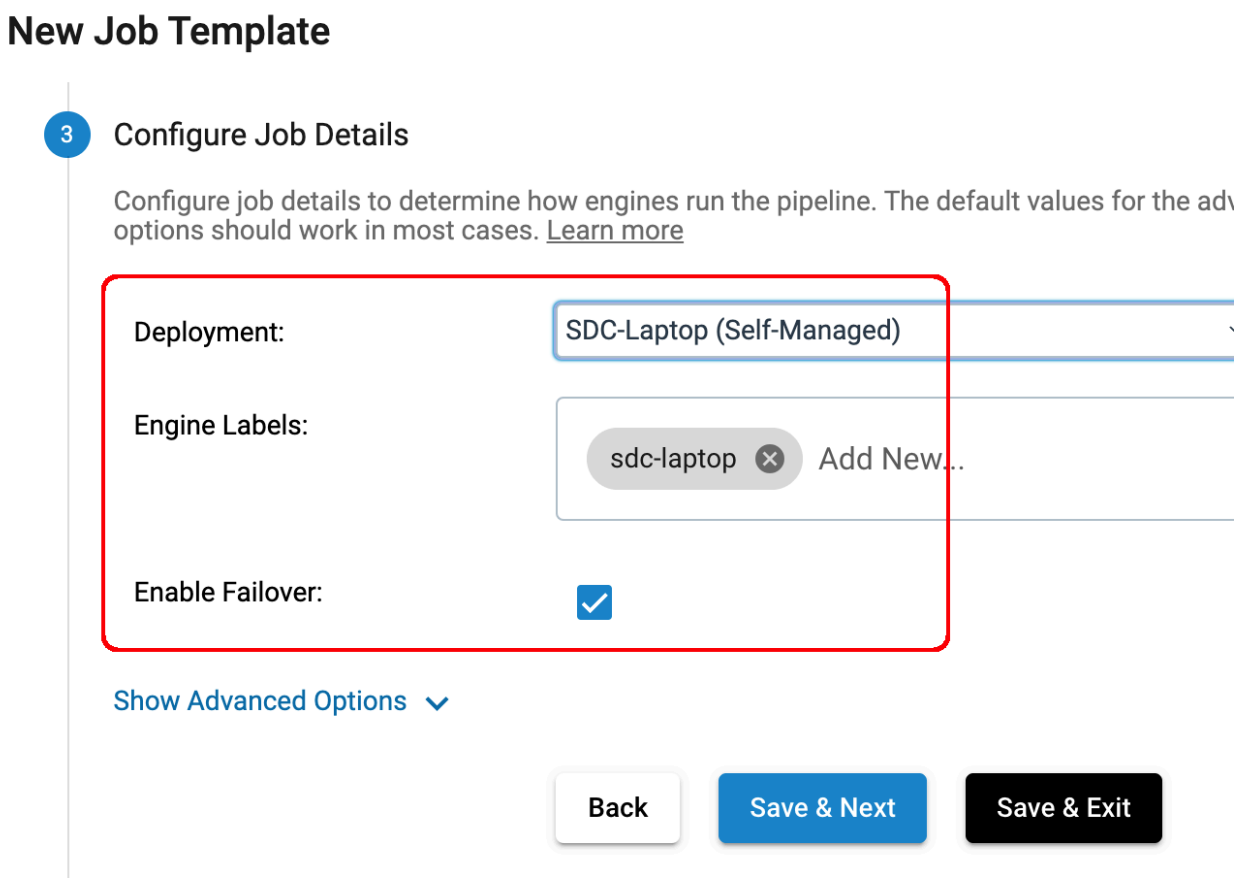
Click Next.

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



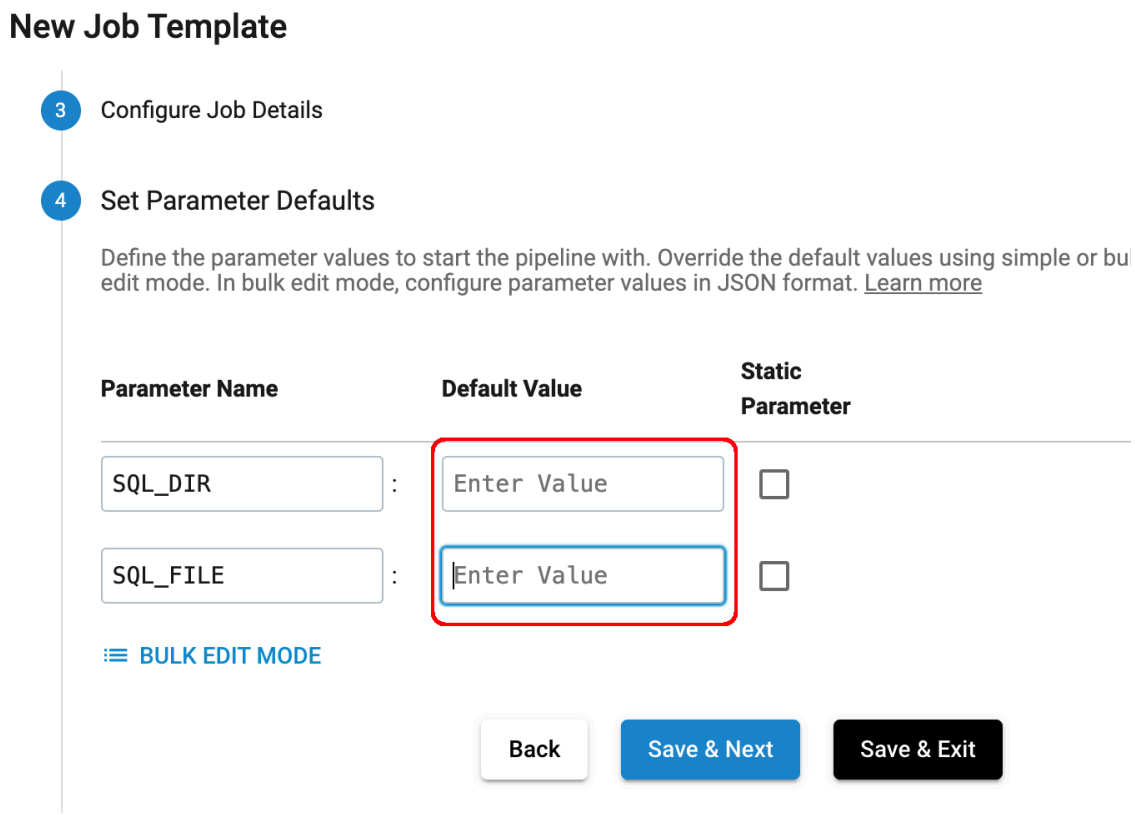
Click Next.

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



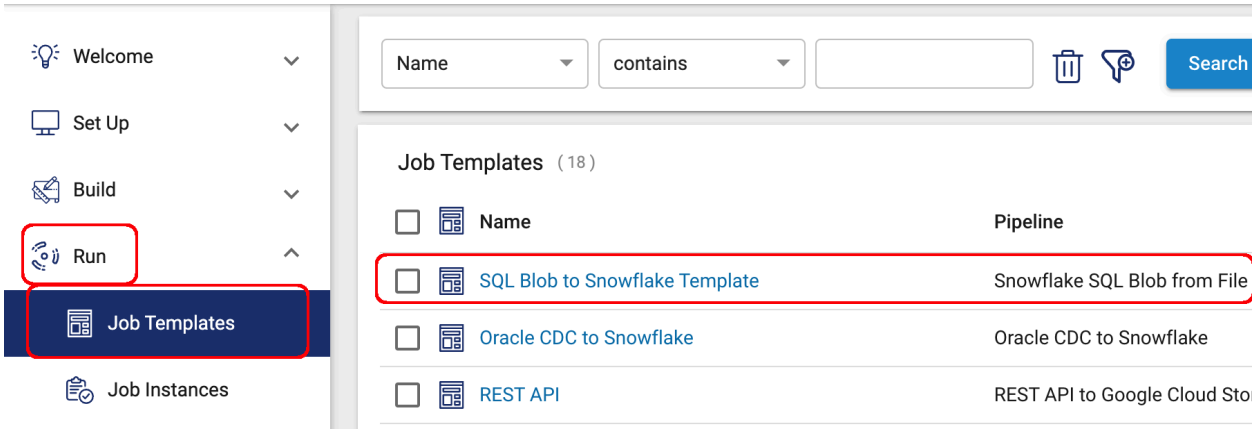
Click Save and Next

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.



Click Save and Next and then Exit.

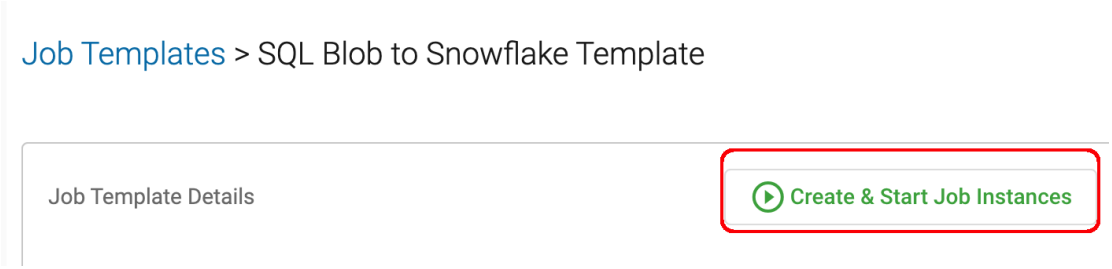
You should see the Job Template that was just created in the Job Template list:



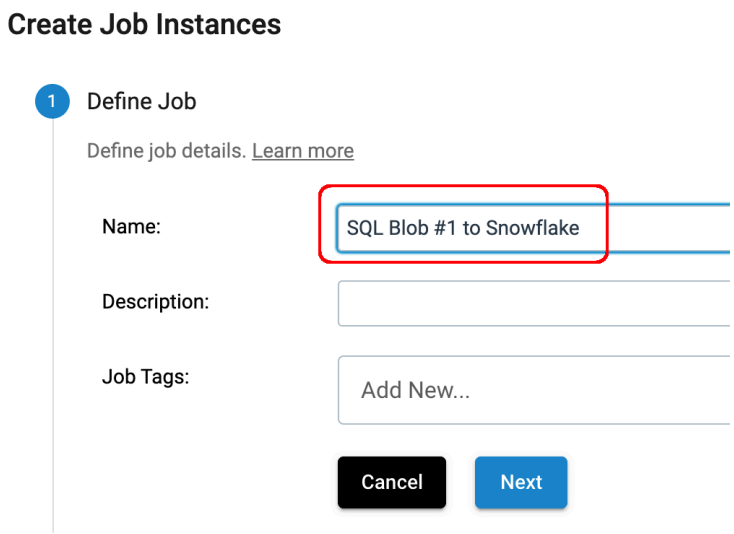
Create and Start a Job Template Instance

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:

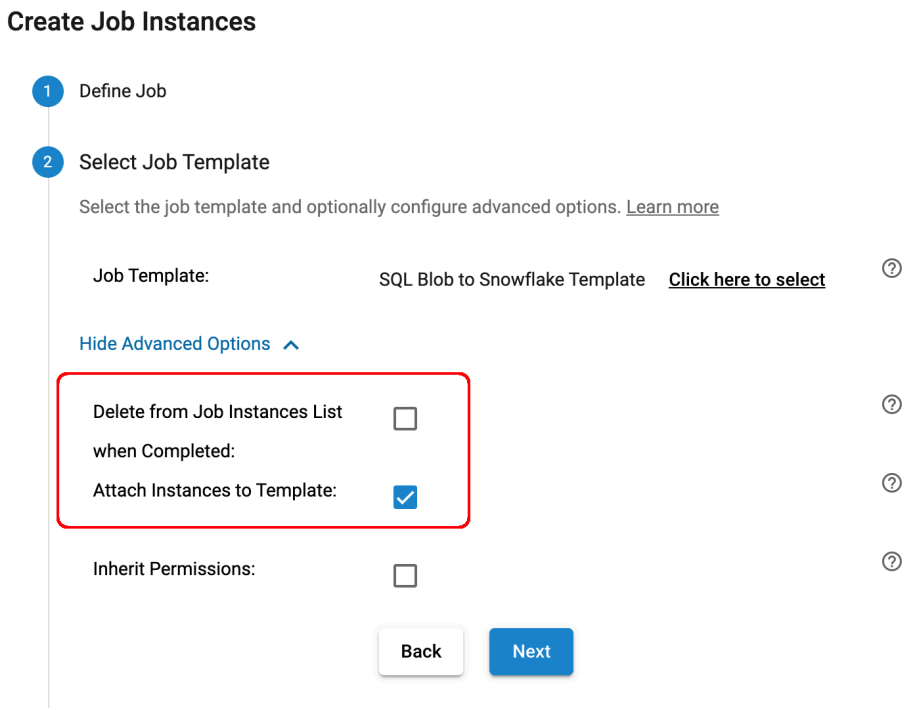


Give the instance a name:



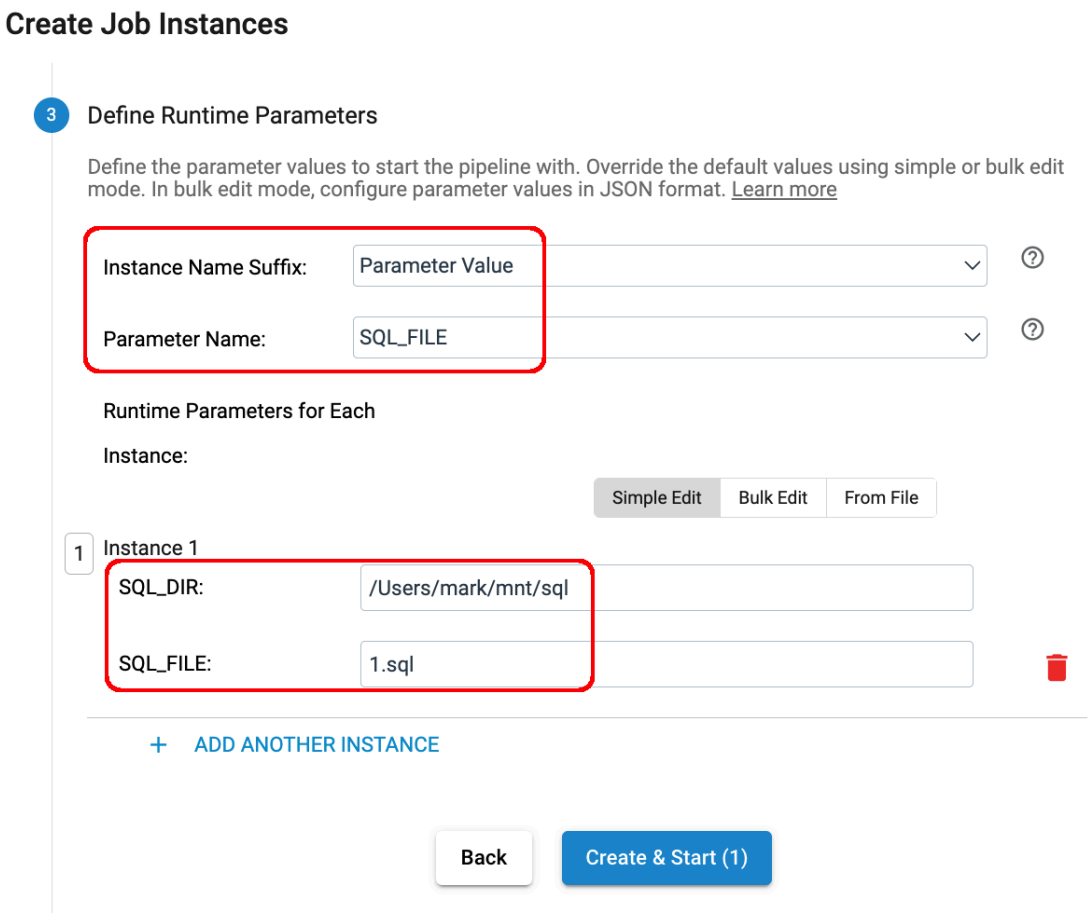
Click Next

Keep these default settings:



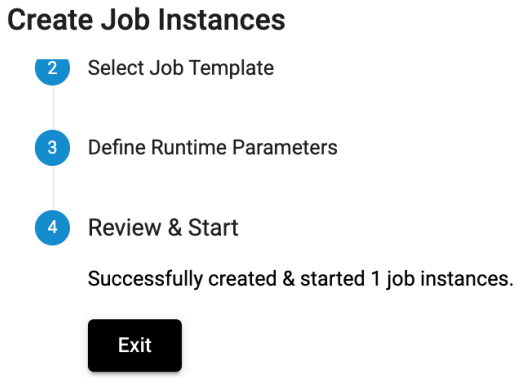
Click Next

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:

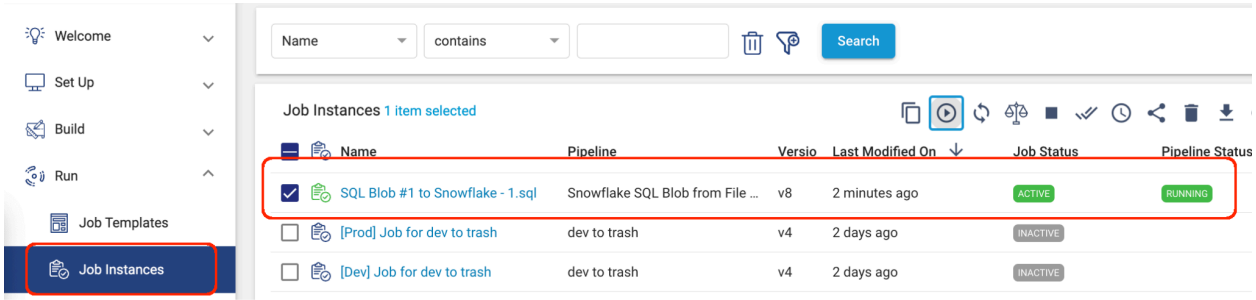


Click the Create and Start button.

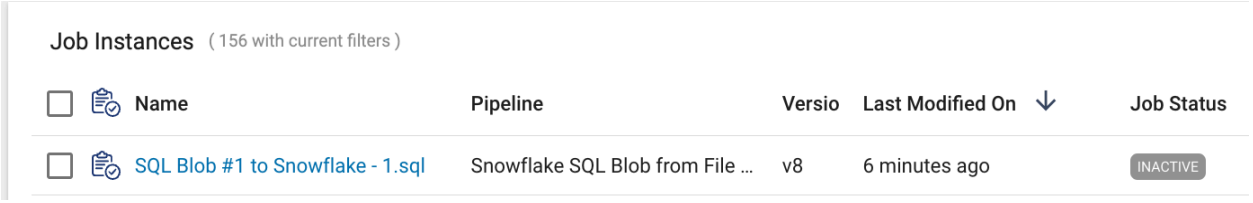
You should see a message like this:



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

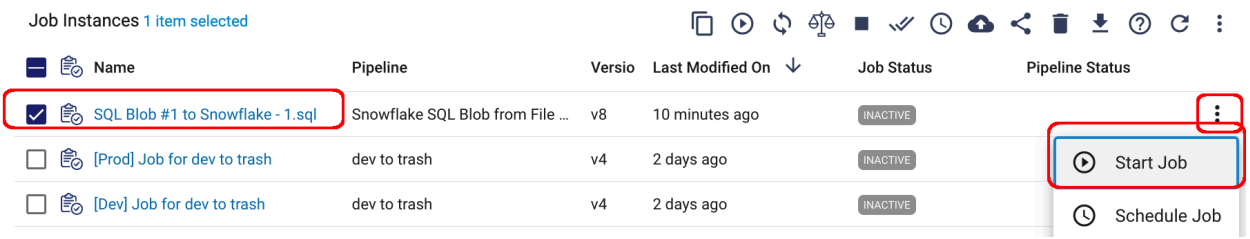


Once the Job completes it will have either an `Inactive` status (if it completed successfully) , or an error status. For example:



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:



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:

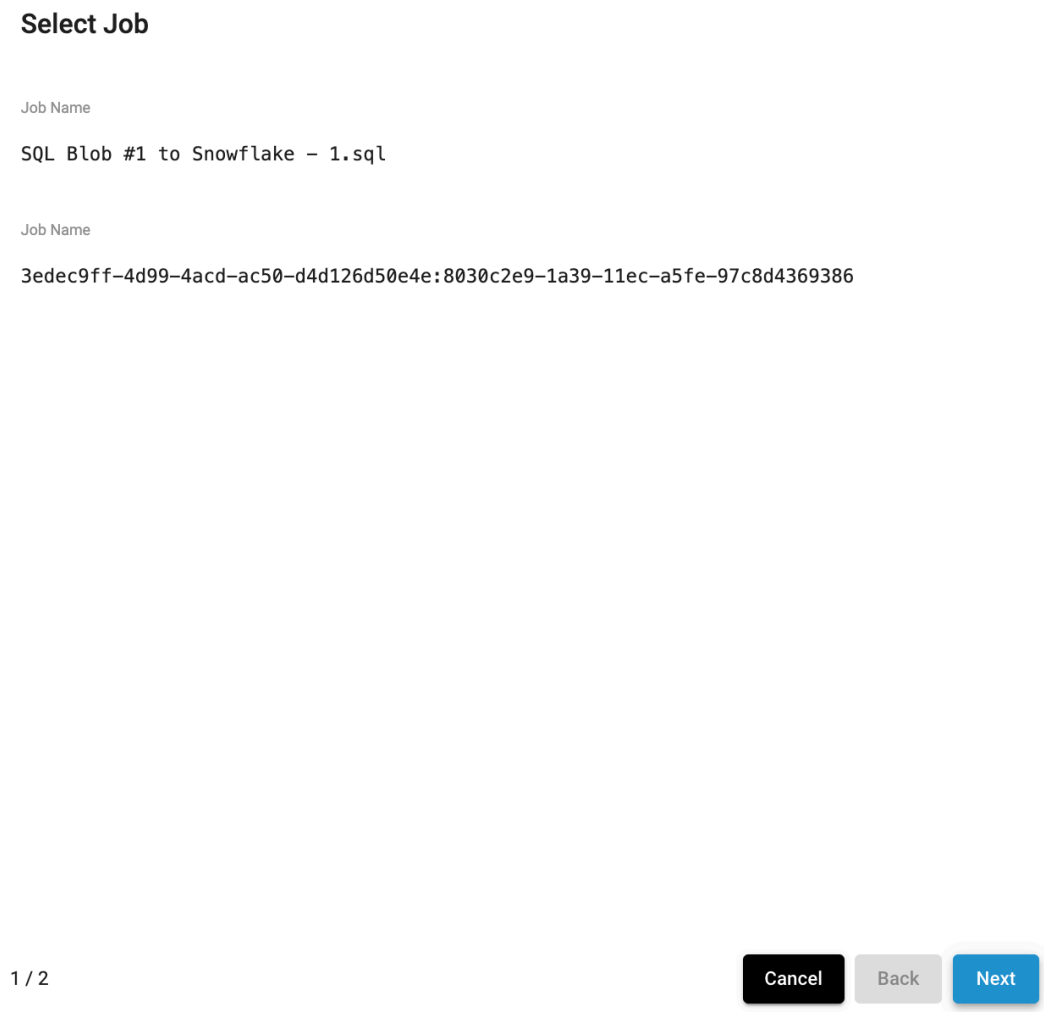
Job Instances (156 with current filters)

Name	Pipeline	Versio	Last Modified On	Job Status	Pipeline Status
SQL Blob #1 to Snowflake - 1.sql	Snowflake SQL Blob from File ...	v8	14 minutes ago	INACTIVE	
[Prod] Job for dev to trash	dev to trash	v4	2 days ago	INACTIVE	
[Dev] Job for dev to trash	dev to trash	v4	2 days ago	INACTIVE	

Start Job

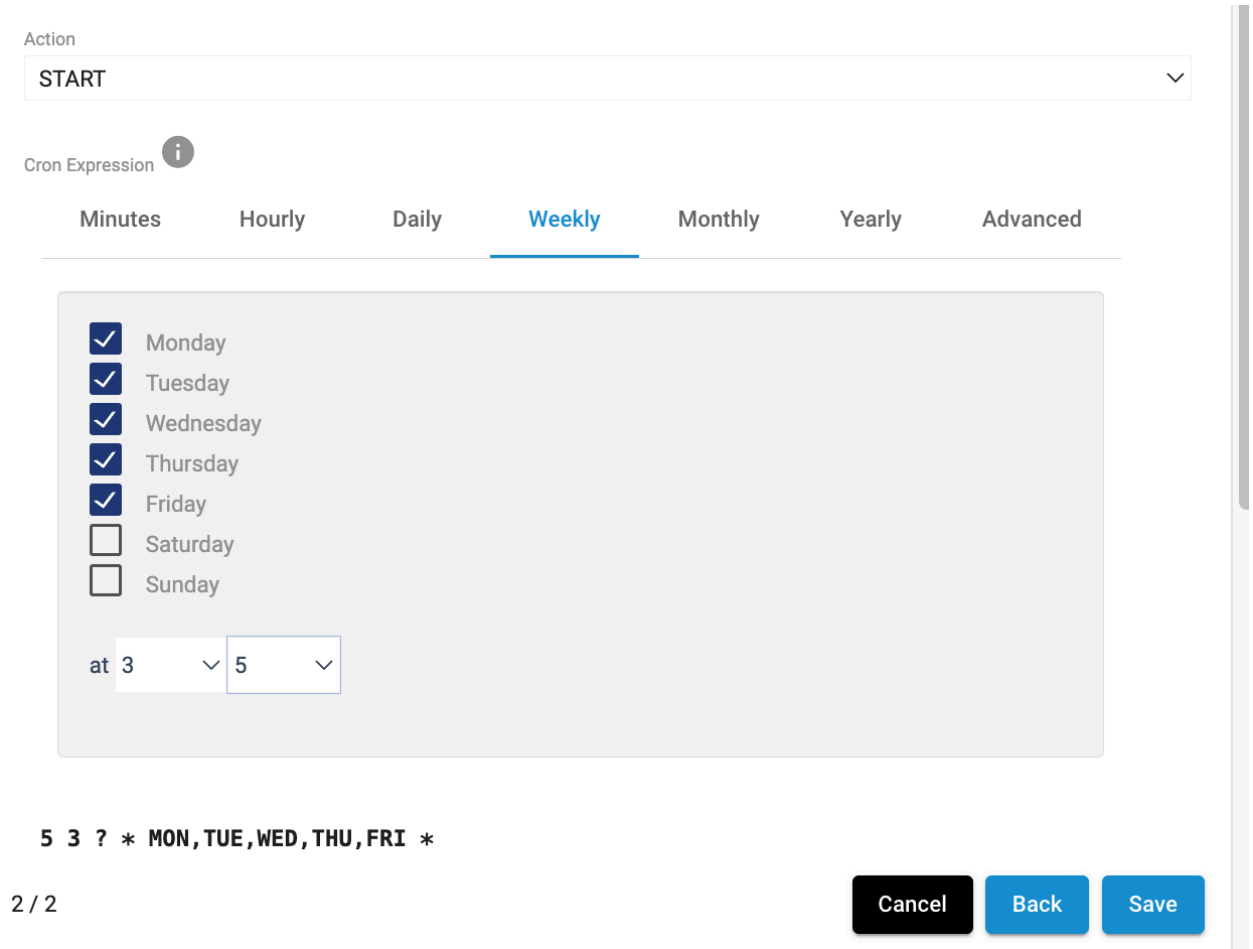
Schedule Job

You will see this screen:



Click the Next button

Set the start time(s) you want:



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