TUTORIAL

Building Data Pipelines in Snowflake: A Beginner-Friendly Guide

Start from scratch and build end-to-end data pipelines in Snowflake by following this guide. We will cover the basics of building automated and efficient Snowflake pipelines.

April 2025 10 min read

Tim Lu
Data Scientist at the New York City Department of Transportation

Data pipelines are a central component of data engineering. They represent the series of steps that move raw data from source systems into a data warehouse, where it can be cleaned, transformed, and made ready for analysis. A well-designed pipeline ensures data flows efficiently, reliably, and securely from ingestion to insight.

Snowflake, a powerful cloud-based data platform, is especially well-suited for building scalable, flexible, and cost-efficient data pipelines. With built-in features for data loading, transformation, automation, and monitoring, Snowflake allows data teams to simplify complex workflows and focus on delivering value from data.

In this tutorial, we'll walk through the process of building a data pipeline in Snowflake—from setting up your environment to automating data transformations using native tools like Streams and Tasks. Some of this will require knowledge of a cloud service's integration process and event management systems. For these, I recommend the documentation of your particular cloud service.

If you're just getting started with Snowflake, consider our Snowflake Foundations skill track to get hands-on and learn to query, model, and manage data.

Understanding Snowflake's Architecture

Snowflake's unique architecture is well-designed to support data pipelines. Snowflake separates compute and storage, which means you can scale each independently.

Data is stored centrally in the cloud storage layer, while computation happens in virtual warehouses, which can be started, stopped, and resized as needed.

This architecture allows multiple users and processes to access the same data without contention, making it ideal for pipeline workloads that involve ingestion, transformation, and analysis.

It also adds a layer of safety where data changes can happen without directly impacting a user's ability to query that data.
For more information and to learn about the cloud layer not discussed here, I recommend this deep dive into Snowflake's architecture.

Key components for Snowflake pipelines

Within each pipeline, we must consider the necessary steps. Typically, you will stage data before loading it into Snowflake. Snowflake itself handles data storage, including tables and their schema, while virtual warehouses provide the compute resources required to execute queries and other data operations.
Here are clear definitions of these essential components used in Snowflake data pipelines:

- **Stages**: Temporary or external storage locations used for loading/unloading data.
- **Virtual Warehouses**:Refers specifically to compute resources provided by Snowflake. Virtual warehouses are essentially Massively Parallel Processing (MPP) compute clusters, composed of multiple Virtual Machines (VMs). Snowflake provisions and manages these resources to process queries and perform data transformations efficiently.
- **Databases**: Logical containers for organizing your data. These often store a collection tables.
- **Schemas**: Subdivisions within a database to group related tables and objects. This defines how the tables are related to one another.
- **Tables**: Where structured data is stored.

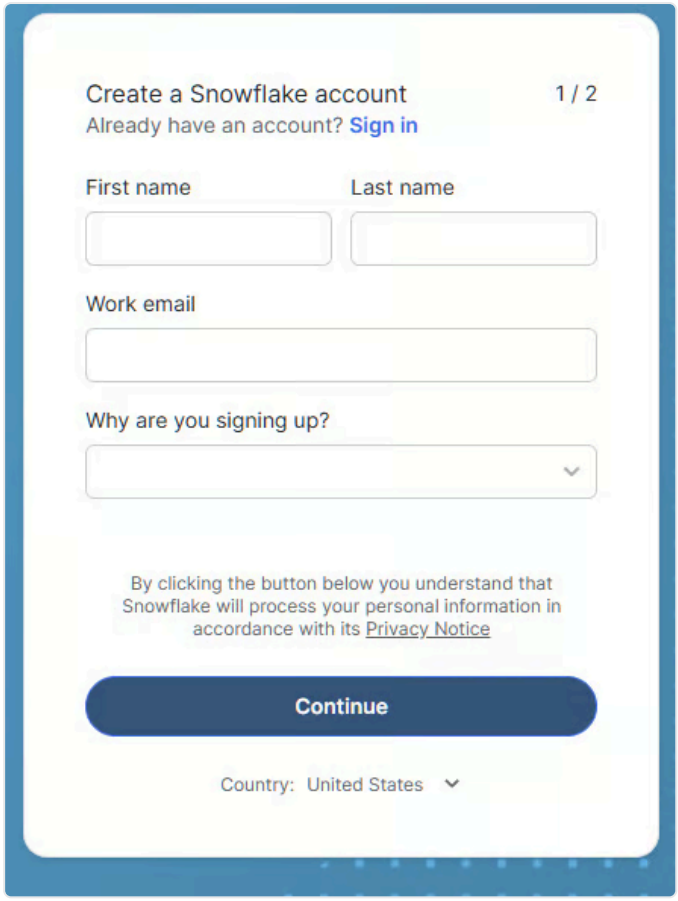
Setting Up Your Snowflake Environment

Let's make sure you have a Snowflake environment ready to go. If you're new to Snowflake, follow these steps to get started:

Create a Snowflake account

If you do not have a Snowflake account, there is a 30-day free trial which includes (at the time of writing) $400 in free usage. This should be enough for us to test a few sample pipelines!

1. Visit Snowflake Trial and sign up for a free account.



Sign-up form for Snowflake Trial

2. For our purposes, we will select the Standard edition instead of the Enterprise edition to save on credit costs in the future if you continue to use Snowflake!
3. Choose your preferred cloud provider (AWS, Azure, or GCP). The examples in this tutorial will use AWS.
4. The automatically chosen data center should be the one nearest to you geographically and should provide the best experience.
5. After account creation, login to the Snowflake Web UI.

Navigating the UI

Once inside, you'll see:

- **Worksheet**: Where you can run SQL queries.
- **Databases**: View and manage databases and schemas.
- **Warehouses**: Monitor and control compute resources.
- **History**: Track previously run queries and tasks

If you want more advice on using Snowflake, follow this Snowflake Foundations course.

Creating necessary resources

This step will cover the creation of a virtual warehouse, database, and schema. Run the following SQL in your worksheet to set up your environment:

```sql
-- set the user role appropriately
USE ROLE accountadmin;

-- Create a virtual warehouse
CREATE WAREHOUSE datacamp_pipeline_wh WITH WAREHOUSE_SIZE = 'XSMALL' AUTO_SU

-- Set our active working warehouse
USE WAREHOUSE datacamp_pipeline_wh;

-- create our Datacamp Sample Database
CREATE OR REPLACE DATABASE datacamp_sample_db;

-- create the Datacamp (Sample) Raw Schema
CREATE OR REPLACE SCHEMA datacamp_sample_db.raw_schema;
```

This SQL code snippet is setting up a Snowflake environment for data processing:1.
• **USE ROLE accountadmin;**: This sets the user role to **accountadmin**, which typically has the highest level of privileges in Snowflake, allowing you to perform administrative tasks.
• 2.
• **CREATE WAREHOUSE datacamp_pipeline_wh WITH WAREHOUSE_SIZE = 'XSMALL' AUTO_SUSPEND = 60;**: This creates a virtual warehouse named **datacamp_pipeline_wh** with a size of 'XSMALL'.
• The **AUTO_SUSPEND = 60** option means the warehouse will automatically suspend (stop) after 60 seconds of inactivity, which helps manage costs.
• 3.
• **USE WAREHOUSE datacamp_pipeline_wh;**: This makes **datacamp_pipeline_wh** the active warehouse for running queries.
• 4.
• **CREATE OR REPLACE DATABASE datacamp_sample_db;**: This creates a new database named **datacamp_sample_db**.
• If a database with the same name already exists, it will be replaced.
• 5.
• **CREATE OR REPLACE SCHEMA datacamp_sample_db.raw_schema;**: This creates a schema named **raw_schema** within the **datacamp_sample_db** database.
• Like the database creation, it will replace any existing schema with the same name.
• Overall, this code is setting up the necessary infrastructure in Snowflake to store and process data, including a virtual warehouse, a database, and a schema.

Was the AI assistant helpful? ✓ Yes ✕ No

Loading Data Into Snowflake

As we discuss in our article about Data Ingestion in Snowflake, Snowflake supports both batch (bulk) and continuous loading. Bulk loading is designed to upload large chunks of data

to your table at a time. This is generally done at scheduled intervals, or automated with Snowpipe's event-based management.
Continuous (stream) loading can be accomplished using things like Snowpipe Streaming. To ingest data, we must create data integrations (stages) with our cloud provider and load the data using `COPY INTO`.

Creating S3 stage and integration

Data ingestion is the first step in any pipeline. In this step, we'll cover the basics of creating a storage integration and creating a stage. To do so, we should imagine we have already connected your Snowflake to AWS with a Snowflake IAM user. If not, refer to the Snowflake AWS configuration guide in the documentation.

```
-- set Role Context
USE ROLE accountadmin;

-- set Warehouse Context
USE WAREHOUSE datacamp_pipeline_wh;

-- this creates the storage integration
-- The STORAGE_AWS_ROLE_ARN will come from the AWS configuration guide steps
-- We will allow access to all locations and block any with sensitive data
CREATE STORAGE INTEGRATION datacamp_aws
  TYPE = EXTERNAL_STAGE
  STORAGE_PROVIDER = 'S3'
  ENABLED = TRUE
  STORAGE_AWS_ROLE_ARN = '<iam_role>'
  STORAGE_ALLOWED_LOCATIONS = ('*')
  STORAGE_BLOCKED_LOCATIONS = ('s3://mybucket/private/path/');

-- Create file format, this is referenced by your stage to easily determine
CREATE FILE FORMAT my_csv_format
TYPE = 'CSV'
FIELD_OPTIONALLY_ENCLOSED_BY = '"';

USE SCHEMA datacamp_sample_db.raw_schema;

--This stage is what ultimately connects your AWS files to your database sch
CREATE STAGE datacamp_s3_stage
  STORAGE_INTEGRATION = datacamp_aws
  URL = 's3://bucket1/path1/'
  FILE_FORMAT = my_csv_format;
```

At the end, you should have successfully created your Snowflake integration with AWS and created the stage. If you want to review AWS basics, the following article is a great introduction to AWS.

Bulk loading with COPY INTO

The command `COPY INTO` is the common method for uploading data to Snowflake. We can either use a locally uploaded file in Snowflake, or use our cloud connections. In the following example, we will use our stage to copy a CSV file.

```
-- Load data using COPY INTO
-- The following will attempt to load all file in the bucket
-- To specify the particular files you can use the FILES or PATTERN settings
COPY INTO datacamp_sample_db.raw_schema.aws_first_upload
FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_csv_format');
```

This SQL snippet is using the **COPY INTO** command to load data from a specified stage into a table in a Snowflake database.
• **COPY INTO datacamp_sample_db.raw_schema.aws_first_upload**: This part specifies the target table where the data will be loaded.
• The table is located in the **datacamp_sample_db** database, under the **raw_schema** schema, and is named **aws_first_upload**.
• **FROM @my_stage**: This indicates the source of the data, which is a named stage (**@my_stage**).
• A stage is a location where data files are stored, often in cloud storage like Amazon S3.
• **FILE_FORMAT = (FORMAT_NAME = 'my_csv_format')**: This specifies the file format to be used when reading the data.
• **my_csv_format** is a predefined file format in Snowflake that likely describes how the CSV files are structured (e.g., delimiter, header presence).
• The code aims to load all files from the specified stage into the target table, using the defined CSV format.
• If you want to load specific files, you can use additional settings like **FILES** or **PATTERN**.

Was the AI assistant helpful? ✓ Yes ✗ No

Automating Loading with Snowpipe

Manually loading every time we get a new file is inefficient and time-consuming as velocity increases. Instead, we can use Snowpipe to load files from our stage. Then, we combine our AWS event notifications to our Snowflake IAM user to trigger the Snowpipe every time.

```
-- Let’ s first create our Snowpipe
CREATE PIPE my_pipe AS
COPY INTO datacamp_sample_db.raw_schema.aws_first_upload
FROM @datacamp_s3_stage
FILE_FORMAT = (FORMAT_NAME = 'my_csv_format');
```

Next, we can use something like Amazon SQS to send notifications to our IAM user whenever our target bucket receives a file.
Finally, we can monitor our Snowpipe using:

```
SELECT * FROM INFORMATION_SCHEMA.PIPE_USAGE_HISTORY;
```

The code snippet is a SQL query that retrieves all records from the **PIPE_USAGE_HISTORY** table within the **INFORMATION_SCHEMA**.
• **SELECT ***: This part of the query selects all columns from the specified table.
• **FROM INFORMATION_SCHEMA.PIPE_USAGE_HISTORY**: This specifies the table from which to retrieve the data.
• **INFORMATION_SCHEMA** is a system database that stores metadata about all other databases on the server, and **PIPE_USAGE_HISTORY** is a specific table within it.
• The code aims to fetch a complete history of how data pipes have been used, which can be useful for monitoring and auditing purposes.

Was the AI assistant helpful? ✓ Yes ✗ No

Now you don't have to worry about uploading data every time a file arrives in your S3 bucket! These cloud notifications will trigger your Snowpipe to load data.

Transforming Data with Streams and Tasks

Snowflake's Streams and Tasks are built-in tools for tracking and transforming changing data. Streams focus on tracking changes to tables where Tasks are utilized for automating the execution of SQL statements.

Introduction to Streams

A Stream captures Change Data Capture (CDC) information—i.e., inserts, updates, and deletes—in a table since the last time it was queried. These are often in the form of Data Manipulation Language (DML) queries. This allows us to see the changes to a table between two different query times.

```
-- Create a stream on a table
CREATE OR REPLACE STREAM my_stream ON TABLE datacamp_sample_db.raw_schema.aw
```

This SQL code snippet is creating a stream named **my_stream** on an existing table **aws_first_upload** located in the **datacamp_sample_db.raw_schema** schema.
• **CREATE OR REPLACE STREAM**: This command is used to create a new stream or replace an existing one with the same name.

• **my_stream**: This is the name of the stream being created.
• **ON TABLE**: This specifies that the stream is being created on a table.
• **datacamp_sample_db.raw_schema.aws_first_upload**: This is the fully qualified name of the table on which the stream is being created.
• It includes the database name (**datacamp_sample_db**), schema name (**raw_schema**), and the table name (**aws_first_upload**).
• The purpose of this code is to set up a stream that will capture changes (like inserts, updates, and deletes) made to the specified table, allowing you to process or analyze these changes in real-time.

Was the AI assistant helpful? ✓ Yes ✗ No

You can then query the stream to get only the changes:

```
SELECT * FROM my_stream WHERE METADATA$ACTION = 'INSERT';
```

The code snippet is a SQL query that retrieves all records from a data stream named **my_stream**.
• It specifically filters the records to only include those where the metadata action is an 'INSERT'.
• Here's a breakdown of the key components: **SELECT ***: This part of the query selects all columns from the records that match the criteria.
• **FROM my_stream**: This specifies the data source, which is a stream named **my_stream**.
• **WHERE METADATA$ACTION = 'INSERT'**: This condition filters the records to only include those where the action in the metadata is 'INSERT'.
• This is useful for focusing on newly inserted records in the stream.
• Overall, the query is used to monitor or analyze new data entries being added to the stream.

Was the AI assistant helpful? ✓ Yes ✗ No

Introduction to Tasks

Tasks allow you to automate SQL execution on a schedule or event trigger. Tasks can either call upon SQL, Javascript, Python, and so on, in order to automate procedures on our tables and pipelines.

```
-- Create a task to process changes and filter data
-- We want changes that occur in the US, for example
-- Schedule to run this task every 5 minutes from our stage
CREATE OR REPLACE TASK process_changes_task
  WAREHOUSE = datacamp_sample_db
SCHEDULE = '5 MINUTE'
AS
  INSERT INTO raw_schema.aws_processed_data
  SELECT * FROM my_stream WHERE location = 'USA';
```

```
-- This will allow us to test the task
EXECUTE TASK process_changes_task;
```

Combining Streams and Tasks

This is the foundation for automated transformation pipelines. As you saw above, Tasks can use Streams to pull in new data to a different table. A general process may look like the following:

- Create a Stream in order to monitor changes to a data table
- Create a Task that pulls data from that stream on a scheduled basis
- With each data pull, transform the data to our desired format
- Load that data into our target table

In this way, we have created parts of a data pipeline. By combining Streams and Tasks, we are able to perform transformations only on newly loaded data. With only Tasks, we would potentially have to process the entirety of our raw tables each time we wanted new data, this can be costly and inefficient use of compute power.

Building an End-to-End Data Pipeline

Let's walk through a simple pipeline use case: ingesting order data, tracking changes, and transforming it for analytics. Since we covered how to execute each step independently, this section covers what the steps might look like together.

Step 1: Create cloud storage integration and stage

First, we have to connect our Snowflake ecosystem to our cloud storage. We used AWS as an example. In Snowflake, we used the `CREATE STORAGE INTEGRATION` and `CREATE STAGE` commands to build the connection between our AWS and our database.

```
-- set Role Context
USE ROLE accountadmin;

-- set Warehouse Context
USE WAREHOUSE datacamp_pipeline_wh;

-- this creates the storage integration
CREATE STORAGE INTEGRATION datacamp_aws
  TYPE = EXTERNAL_STAGE
  STORAGE_PROVIDER = 'S3'
  ENABLED = TRUE
  STORAGE_AWS_ROLE_ARN = '<iam_role>'
  STORAGE_ALLOWED_LOCATIONS = ('*')
  STORAGE_BLOCKED_LOCATIONS = ('s3://mybucket/private/path/');
```

```
-- Create file format, this is referenced by your stage to easily determine
CREATE FILE FORMAT my_csv_format
TYPE = 'CSV'
FIELD_OPTIONALLY_ENCLOSED_BY = '"';

USE SCHEMA datacamp_sample_db.raw_schema;

--This stage is what ultimately connects your AWS files to your database sch
CREATE STAGE datacamp_s3_stage
  STORAGE_INTEGRATION = datacamp_aws
  URL = 's3://bucket1/path1/'
  FILE_FORMAT = my_csv_format;
```

Step 2: Ingest data with Snowpipe

In this second step, we create a Snowpipe that will run the `COPY INTO` command into our raw data tables.

```
-- Let’ s first create our SnowpipeCREATE PIPE my_pipe AS

COPY INTO datacamp_sample_db.raw_schema.aws_first_upload

FROM @datacamp_s3_stage

FILE_FORMAT = (FORMAT_NAME = 'my_csv_format');
```

Step 3: Use Streams to capture changes

We then use a stream on this raw data table in order to track changes.

```
-- Create a stream on a table
CREATE OR REPLACE STREAM my_stream ON TABLE datacamp_sample_db.raw_schema.aw
```

This SQL code snippet is creating a stream named **my_stream** on an existing table **aws_first_upload** located in the **datacamp_sample_db.raw_schema** schema.

- **CREATE OR REPLACE STREAM**: This command is used to create a new stream or replace an existing one with the same name.
- **my_stream**: This is the name of the stream being created.
- **ON TABLE**: This specifies that the stream is being created on a table.
- **datacamp_sample_db.raw_schema.aws_first_upload**: This is the fully qualified name of the table on which the stream is being created.

• It includes the database name (**datacamp_sample_db**), schema name (**raw_schema**), and the table name (**aws_first_upload**).
• The purpose of this code is to set up a stream that will capture changes (like inserts, updates, and deletes) made to the specified table, allowing you to process or analyze these changes in real-time.

Was the AI assistant helpful? ✓ Yes ✗ No

Step 4: Automate transformations with Tasks

Next, we create a Task that transforms and loads this newly loaded data to our processed version of the table. This will trigger on a schedule to check for new data. If there is no new data, the Task won't run.

```
-- Create a task to process changes and filter data
-- We want changes that occur in the US, for example
-- Schedule to run this task every 5 minutes from our stage
CREATE OR REPLACE TASK process_changes_task
  WAREHOUSE = datacamp_sample_db
SCHEDULE = '5 MINUTE'
AS
  INSERT INTO raw_schema.aws_processed_data
  SELECT * FROM my_stream WHERE location = 'USA';

-- This will allow us to test the task
EXECUTE TASK process_changes_task;
```

Step 5: Monitor and manage

Finally, we want to make sure our pipelines are working as intended. Make sure to use the `SHOW PIPES` and `SHOW TASKS` commands to verify these are properly set up and running. Then use metadata tables from `INFORMATION_SCHEMA` to track data loading performance and errors.

Best Practices for Data Pipeline Development in Snowflake

As always, we want to ensure that we are operating with best practices in our development. Things to consider are optimizing for performance and cost, ensuring data quality and reliability, and implementing the appropriate levels of security and access.

Optimize for performance and cost

Our goal should be to always operate our pipelines as efficiently as possible with as little cost. This way, we are minimizing our impact on servers and providing data in as timely a

manner as possible. Here are some tips:

- **Use auto-suspend on warehouses to avoid idle charges**. This cuts down on compute costs by having warehouses operating, since they charge by time.
- **Load compressed files and batch them into optimal sizes**. Files that are too small take up too much network overhead by needing to generate connections to load each file. Files that are too large may take up too much bandwidth and risk failing. Snowflake recommends deltas of 100-250 MB in size.
- **Schedule transformations during low-usage windows**. This means you are not competing for resources.

Ensure data quality and reliability

Data quality and reliability are critical for every data engineer to consider. You want to make sure the data being loaded is usable without errors. You also want to make sure your data pipeline won't inadvertently cause data to drop when there are issues. Here are some tips for data quality management:

- **Validate data before and after ingestion**. This can be done with periodic checks of staging tables and alerts.
- **Use streams and staging tables to isolate and fix problematic records**. Since only changed data gets pulled in, we can minimize the scope of our issues using streams.
- **Apply unique constraints or deduplication logic where needed**. This can prevent things like `NULL` data where we cannot have any or pre-dedupe in the event of overlapping data pulls.

Implement security and access controls

Finally, it is important to ensure proper access and security to your data. Some tables can hold sensitive information or should not be altered except by a select group. Follow this advice to properly manage your security and access in Snowflake:

- Use **roles and grants** to restrict access.
- Protect sensitive data with **column-level masking** or **row access policies**.
- Rotate storage credentials and monitor stage usage.

Conclusion

Building data pipelines in Snowflake is a streamlined and scalable process that is perfect for beginners stepping into the data engineering world. With its intuitive architecture, robust ingestion options, and built-in automation tools like Streams and Tasks, Snowflake empowers data teams to move quickly from raw data to actionable insights.
By following this guide, you now have the foundational knowledge to set up your own Snowflake pipeline—handling ingestion, transformation, and orchestration, all within a single

platform. For more information on Snowflake, check out the following guides:

- Snowflake for Beginners
- Introduction to Snowflake Course
- How to Learn Snowflake
- Data Modeling in Snowflake
- Best Snowflake Certification 2025

TOPICS