Data Streaming at Scale: Databricks and Snowflake

Ryan Chynoweth

19 min read · May 10, 2023

Special thanks to co-author Kyle Hale.

The following is the outline of the blog which highlights the different areas of stream data processing.

- Introduction
- Data Ingestion
  - Ingesting Data with Databricks
  - Ingesting Data with Snowflake
- Stream Data Processing
  - Databricks Stream Processing
  - Snowflake Stream Processing
- Data Serving
- Conclusion

Introduction

Processing data in real-time is a critical ability for organizations in all industries and sizes. Streaming data is unbounded and has no predetermined beginning or end as it is a series of events that arrive at a system. Batch data has set input boundaries to compute results a single time. The vast majority of data is generated in a continuous manner, therefore, processing data as a stream is critical to meeting evolving business requirements. Data streaming makes the following possible:

- **Notifications and alerting:** given a series of events an alert should be triggered
- **Real-time reporting:** monitor platform usage, uptime, or new product features
- **Incremental ETL:** process incremental data instead of large batch jobs
- **Serving data in real-time:** compute and serve data between applications
- **Real-time decision making:** take action before the transactions are confirmed
- **Online machine learning:** use machine learning in real-time compared to hard coded business rules

Processing streaming data can be complex and difficult. At some point in the development process the engineer creating the system has a choice to make: build the system to handle the complexity or push the complexity onto the user. As described by Ralph Ammer, there is a fine balance between a system with a single button and overloading the user with too many options.

In this blog we'll contrast the streaming solutions provided by Databricks and Snowflake. Ultimately you will discover that Databricks offers a simple

approach that took years of engineering to create. Snowflake's streaming capabilities are limited and do not provide the functionality, simplicity, and observability operate at scale. It is clear that the current features available on Snowflake do not permit streaming pipelines.

We will limit the scope of this analysis to features that are publicly available as of this blog's publication date and each product's ability to natively support workloads without additional third-party tooling or excessive custom development. In short, the streaming capabilities for both products can be found in the table below.

	Databricks	Snowflake
Relative Performance	Sub-Second	Minutes
Message Bus Integration	Direct Access	Requires separate compute (non-Snowflake compute)
Unified Batch and Streaming	Yes	No
Streaming Trigger Types	Continuous, Once, Intervals, Recurring, AvailableNow	Once, Intervals
Data Residency	In-memory or Open Source Table Format in customer account	Proprietary persisted data in vendor account
Supported Languages	SQL, Python, Scala	SQL, Scala & Python are translated to SQL
Supported Workloads	Data Engineering, Data Science, Machine Learning, and Business Intelligence	Data Engineering, Business Intelligence
Data Source Connectors	Large open ecosystem of connectors including SQL Databases, Key-Value Databases, Cloud Object Stores, APIs, Message Buses	Proprietary connectors to Cloud Object Stores

Feature Overview Table

To successfully implement a streaming pipeline, a product must be performant, simple, and reliable. Databricks provides the ability to choose the level of simplicity desired as there are two ways to implement streaming solutions on Databricks: Spark Structured Streaming via Databricks jobs and Delta Live Tables. Databricks runs millions of streaming jobs on a weekly basis and has a robust set of connectors to orchestrate data movement within and outside of Databricks. Databricks has the ability to monitor solutions at a production level while handling dependencies, infrastructure, and data quality. Streaming jobs have the ability to run 24/7 or set on a triggered cadence to satisfy business requirements.

Snowflake is a cloud data warehouse and that is the foundational core of the product. Snowflake requires third party applications to ingest data from streaming sources. The landing data in the Snowflake table must be a copy of source data which adds additional latency. Table to table streaming in Snowflake is an isolated batch process that can be scheduled at most once a minute further increasing latency. Once data is prepped and ready to share, there are limited capabilities to publish data to external systems. Ultimately, complexity and latency will be a burden to the engineer and make it difficult to meet the required SLAs. Snowflake is not a viable streaming solution.

Data Ingestion

Enterprise message buses are the dominant data source for streaming solutions, the ability to connect and process data from these systems is critical. They are used to ingest unbounded and continuous data and make the data available to multiple consumers. Apache Kafka is widely used for this exact purpose, therefore, we will use it as an example source for our evaluation.

Databricks provides a low latency, high throughput data source connector that allows engineers to connect directly to Kafka and read the data into memory. In Databricks, data can be transformed in-memory and written to external systems or persisted to tables for further analysis. This allows a robust set of features enabling engineers to build scalable solutions that integrate into your broader IT ecosystem.

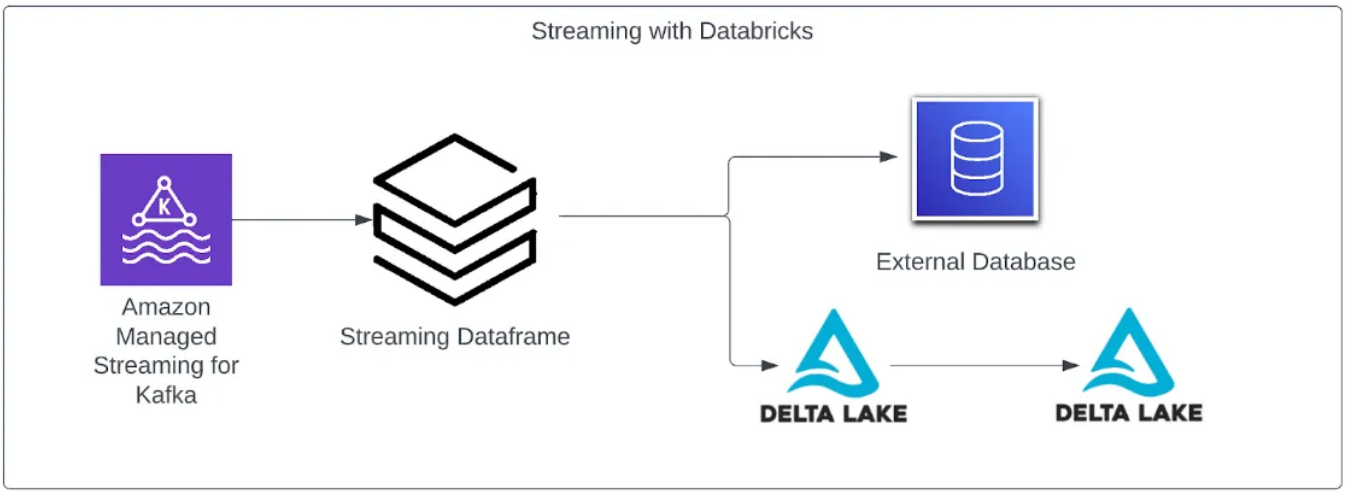
Snowflake does not have the capability to connect to Kafka directly, as it relies heavily on external applications. Snowflake requires engineers to deploy an external application that does not run on Snowflake compute to connect to Kafka and write to a Snowflake table. Data is required to be persisted multiple times and duplicates the data that is already available in Kafka as a Snowflake table. Ingesting streaming data using Snowflake adds unnecessary latency which in turn increases the amount of compute required to process your data. Therefore, not only is data processed slower but each batch of data needs more resources.

Ingesting Data with Databricks

Databricks and Apache Spark have been implementing streaming solutions in **production** since Spark 2.0 (i.e. 2016). In that time, the number of Databricks streaming jobs has grown to over 4 million a week! When you throw in open-source Apache Spark and other cloud hosted Spark offerings the number of streaming workloads on Spark is an impressive testament to its abilities.

As one of many data source options, Databricks can natively stream data from a Kafka topic. Engineers have the choice to keep data in memory for low latency processing, persist the data to Delta tables (which can also be used as a streaming source), or both! Additionally, Databricks is built on top of open-source projects, allowing customers to obtain a level of technical

portability not possible with proprietary tooling. Below is a diagram of streaming data from Kafka with Databricks, and optionally writing to external systems for in application alerts or persisting the data to Delta for near real-time analytics.



Databricks Spark Structured Streaming Architecture with Kafka

As data is streaming into Kafka, Databricks can connect directly and process this data in-memory without having to persist to an intermediate storage account. This allows for much faster and scalable streaming solutions when compared to Snowflake's batch process that constantly requires rewriting of data. Please reference this repository for a Databricks notebook example, however, below is the code to read data from Kafka.

```
kafka = (spark.readStream
 .format("kafka")
 .option("maxBytesPerTrigger", 200000)
 .option("kafka.bootstrap.servers", kafka_bootstrap_servers_plaintext )
 .option("subscribe", topic )
 .option("startingOffsets", "earliest" )
 .load())
```

```
read_stream = kafka.select(col("key").cast("string").alias("eventId"), from_json
display(read_stream.select("eventId", "json.action", "json.time"))
```

	eventId	action	time
1	f5209f1e143846edae57a0629ce6f2bf	Close	1469596778
2	523c3c66f5a54aafb65a9e88a3354406	Open	1469539208
3	f3f62d2efffc44739a6550e6cc67a9f7	Close	1469618338
4	bf7446a3e25542d8b7e691d5bd243f40	Open	1469596779
5	bbca3c3229d2406f82f7f43b1761f8d5	Open	1469618338
6	c4f6b28c853643d19287c258e4468bab	Open	1469539209

Results from the above code

With Databricks, reading data from Kafka is simple and performant and requires very few lines of code to easily transform semi-structured data into columns **prior** to writing to a table. The ability to transform data while ingesting may seem unimportant but this is table stakes for a streaming solution — this is a streaming capability that Snowflake does not possess. The Databricks platform simply manages and hosts the connection to Kafka so the engineer can focus on providing maximum value to the organization.

Ingesting Data with Snowflake

Snowpipe

Data pipelines within Snowflake are used to automate steps within your ETL process and are used to optimize “continuous data loads”. The idea that Snowflake Snowpipe is used for streaming data is simply untrue, as it is a tool to batch load data from cloud storage into a table on a recurring basis. Snowflake recently released an entirely new Snowpipe Streaming API which

shows that the current feature set does not allow for processing streaming data at scale.

Processing data in Snowflake simply requires a lot of reads and writes to tables. Data pipelines must load data from a Snowflake stage, which means unloading and loading data with Snowflake is more complex and expensive than it should be.

At its core, Snowpipe is a tool to copy data into Snowflake from cloud storage. Snowpipe allows users to split data loads into multiple transactions depending on the data size, plus compute resources can be supplied by Snowflake via a serverless offering so users do not need to provision a virtual warehouse. Data loading events can be triggered using cloud messaging or the REST APIs.

Databricks has a similar feature that we call Auto Loader. Auto loader enables developers to create a Spark Structured Streaming pipeline with cloud files as a data source. This pipeline will track which files have been loaded, continuously load files as they are produced as a stream, and can optionally be turned into a batch process with a single line of code. Auto Loader can be used in Scala, Python, and SQL, and supports many file types with schema inference. While Snowpipe stands for a large portion of Snowflake's streaming capabilities, Auto Loader is just a small piece of the solution.

Snowpipe Streaming

Snowflake recently promoted their Snowpipe Streaming API to public preview status, which allows for low-latency streaming ingestion into Snowflake. While performance has improved, the new API suffers from much of the complexity that the legacy Snowpipe API faces. It is worth

noting that, according to the Snowpipe Streaming Announcement, engineers can expect "sub-5 second median latencies" which continues to lag in performance when compared to Databricks and only applies to ingestion, not data processing.

The new API streams data into Snowflake and writes it to a table as a BDEC file type, which allows data to be available quickly. However, BDEC files need to be rewritten ("migrated") to the native Snowflake file format, FDN. This is essentially a workaround for Snowflake to stream data quickly into a table, then charge the customer additional compute to rewrite the data.

This begs the questions — what is the limitation of the existing FDN file type in Snowflake that does not support writing data as a stream? Will this limitation continue to be an issue as Snowflake looks to improve streaming within their product?

Snowpipe Streaming is enabled by a Java SDK, which phData wrote an excellent article on how to use. In the article they state, "building a whole new application might sound like a lot of work", which is an accurate description of building a streaming solution on Snowflake. The quote in the previous sentence is related to the fact that you can simply upgrade your existing Kafka connector application to a new version, which reinforces the idea there is no new functionality from an architectural perspective. The service still requires an external application that does not run on Snowflake which increases complexity, latency, and costs.

Snowpipe Streaming appears to improve Snowflake's ability to ingest streaming data, while the complexity of the system as a whole remains high. Once data is in Snowflake engineers are back to batch processing of data.

What is the point of ingesting data as a stream if the compute engine cannot operate on the data as a stream? While performance for ingestion has improved, there are no new capabilities. Just a new API.

Snowflake Kafka Connector

Apache Kafka is a popular open-source tool used to ingest streaming data, therefore, it is logical that Snowflake needs to integrate with this system. Kafka Connect is a declarative integration framework for Kafka that allows data solution providers to create specialized connectors for their product. It is at the core of Snowflake's "streaming" capabilities.

The Kafka Connector for Snowflake leverages Snowpipe to load the data into a target table. Once data is in Snowflake's internal storage, an extremely low latency process is turned into a batch process in Snowflake where you can only use SQL to transform data. Even Snowpark is simply an API that "provides programming language constructs for building SQL statements".

Engineers must deploy a separate software application to ingest data from Kafka which can be shared by multiple topics to load data in parallel. Users should consider load balancing implications as it may require multiple deployments of the application, further increasing complexity.

Each Kafka topic corresponds to a single table and that table contains two columns: `RECORD_CONTENT` and `RECORD_METADATA`. Engineers can specify more columns that will be set to `null` upon insert and you must do further ETL processing to provide values as transformations in transit are not supported. Since transforming data when loading is not supported, the target table in Snowflake results in a copy of the Kafka source which introduces unneeded I/O and extra storage.

Row	RECORD_METADATA	RECORD_CONTENT
1	{ "CreateTime": 1660854152701, "offset": 1, "partition": 0, "topic": "kafka_test_messages" }	null
2	{ "CreateTime": 1660854909038, "offset": 7, "partition": 0, "topic": "kafka_test_messages" }	{ "value": "sample message" }
3	{ "CreateTime": 1660854888170, "offset": 5, "partition": 0, "topic": "kafka_test_messages" }	{ "value": "this is a sample message" }

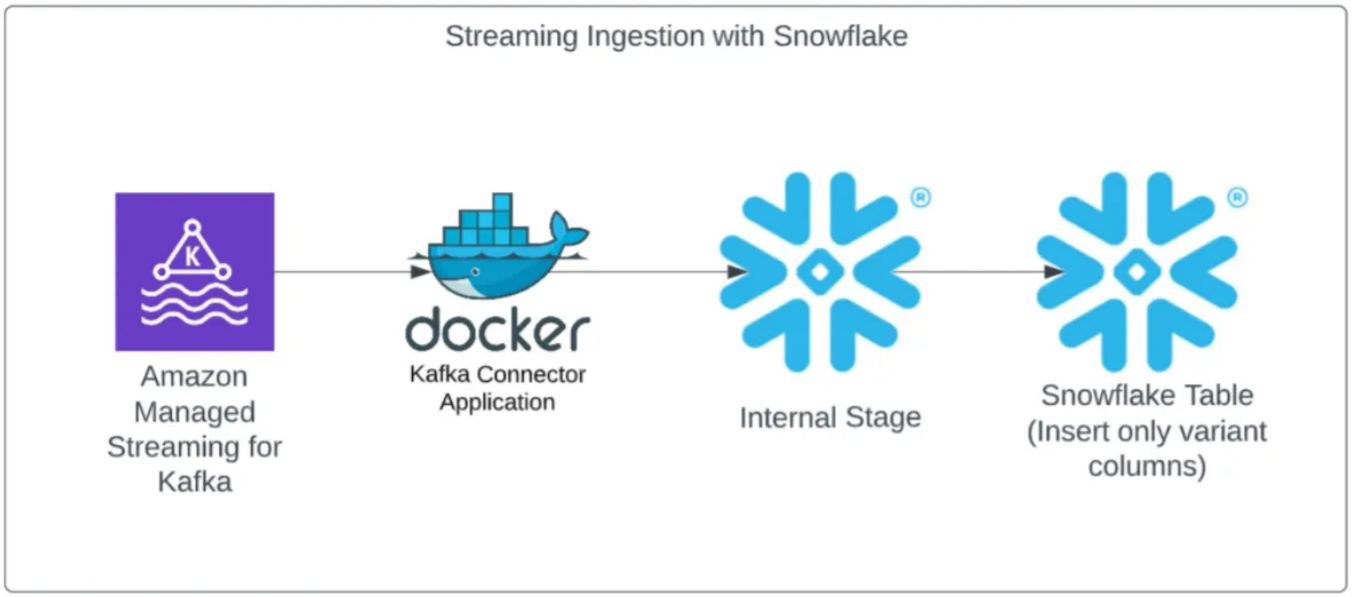
Example Snowflake Target Table

Imagine having many topics — it's a crazy amount of overhead to simply load raw data into Snowflake! We are not even talking about business value at this point.

On top of the requirements, engineers **must persist data**! The data must be written to a table and that table is essentially a duplicate of what is available in Kafka. Alternatively, in Databricks this is not required. If Databricks' customers require low latency data processing, then they can keep the data in memory to quickly publish to external systems.

The Kafka connector is installed on compute that is **not** Snowflake infrastructure, therefore, it is not truly a service offered by Snowflake at all. Customers must manage Kafka connect themselves or pay another provider for a managed service, again absorbing additional costs and complexity.

Below is a diagram describing how streaming data can be ingested into Snowflake as an insert only variant typed table. The need for an external Kafka Connector application adds to the complexity, latency, and unneeded costs to your streaming application.



Streaming Ingestion with Snowflake

Please note that the Docker logo is present because of the separate application that is required. For an example on how to configure and deploy the Kafka connector on Docker, please refer to the supporting repository.

Stream Data Processing

Databricks has two production ready products for building streaming applications which gives users flexibility to choose how they want to develop. Streaming solutions on Databricks abstract the complexity of managing continuous data at scale while still providing the technical knobs to tune performance for a particular use case. Data on Databricks can be held in-memory for low latency data processing or persisted as tables. Tables in Databricks are extremely robust and integrate extremely well with Spark Structured Streaming. Delta Tables can not only act as streaming sources and sinks, but batch sources and sinks — unifying different workloads.

Snowflake users must combine a number of different SQL objects to build pipelines. Pipelines are able to collect isolated change data from source tables, transform the data, and write the data to an output table. Since each

execution is done in isolation there are limitations on the type of transformations that can be done in a streaming manner. Pipelines are executed using the task scheduling system which has severe limitations on how frequent the task can be triggered. Solutions built on Snowflake become extremely difficult to manage due to proliferation of objects and do not have the observability metrics required to operate at scale. Due to query scheduling Snowflake pipelines have difficulty keeping up with data that is generated in real-time.

Databricks Stream Processing

Spark Structured Streaming

Spark Structured Streaming is a great solution for both analytical and operational workloads. When working with Spark Structured Streaming on Databricks, data resides in memory and can be written to a table in Databricks. Optionally, users can avoid persisting data in Databricks altogether and apply transformations in memory then write to an external system e.g. an application database for personalized recommendations.

Once data is persisted as a table in Databricks, users have the ability to stream data changes from table to table using the Delta Lake streaming data source and sink. Please reference the code below for table to table streaming in Databricks. Yes, it is really that simple.

```
# table streaming sink
( read_stream.writeStream
     .option("checkpointLocation", "/tmp/delta/events/_checkpoints/")
     .toTable("events")
  )
```

```
# table streaming source
spark.readStream.table("events")
```

The code above is deceptively simple. Under the hood Spark does many complex tasks such as ensuring exactly-once and fault-tolerant processing through the use of checkpointing. Structured streaming sources, sinks, and the execution engine can track the exact progress so that it can seamlessly handle any kind of failure by restarting the stream and beginning at the last saved checkpoint.

Performance and simplicity is the top priority for Databricks. Enabling customers to build streaming pipelines as quickly as possible is important to improve return on investment for particular solutions. Building fast pipelines quickly means users realize their gains sooner.

Delta Live Tables

Delta Live Tables (DLT) is a framework built on top of Spark Structured Streaming that is used to develop reliable, maintainable, and testable data pipelines. Structured streaming is still an excellent tool for production workloads, however, DLT makes streaming on Databricks even more simple, allowing engineers to create declarative pipelines in SQL and Python. Ultimately, engineers can define transformations and allow Databricks to automatically manage task orchestration, cluster management, monitoring, data quality, and error handling.

Pipelines in Delta Live Tables are the unit of execution and are represented as a directed acyclic graph (DAG) to monitor the relationships between datasets. Pipelines are created using Databricks notebooks written in Python or SQL, and each pipeline can have many associated notebooks that are version controlled using Databricks Repos.

When creating pipelines engineers can create *datasets* that are represented as either views or tables which can be *live* or *streaming live*. A live dataset represents the results of the query like a materialized view. A streaming live table only processes new data since the last pipeline execution.

A Delta Live Table pipeline can stream data from any source that Spark Structured Streaming can connect to which includes the ability to stream between tables. As an example, if you want to stream data from a set of JSON files you can do the following:

```
CREATE STREAMING LIVE TABLE my_table
AS SELECT * FROM cloud_files("/path/to/json/files", "json")
```

Delta Live Tables has emerged as the preferred way to do ETL because it opens Spark Structured Streaming to a wider audience and automates most of the manual production activities required at scale. Databricks absorbs the complexity so that the user focuses on providing the maximum amount of value to the organization.

Snowflake Stream Processing

Snowflake Streams

A Snowflake Stream is an object that allows users to process changed data so that only new records are selected from a table. Therefore, Snowflake Streams are not necessarily a stream process, but a way for Snowflake to manage offsets and obtain updated records.

Streams are a critical functionality required to *simulate* a streaming process on Snowflake. One can write a task that collects the changed data from a source table, transform, and merge that data into a target table. The task can then be triggered on demand (once) or a recurring basis (interval), however, in both scenarios it is a batch process with high latency. Spark structured streaming supports both one time and interval based streaming models, but also supports continuous and recurring (default) processing on data streams.

SQL engineers can read from a Snowflake Stream once and write to many target tables using multi-table inserts, which means that for more complex logic you will need to leverage temporary tables and are limited to insert only operations. Spark Structured Streaming does this as well by utilizing a foreach batch function that supports robust data transformation capabilities. In Databricks we can easily merge source data into multiple target tables, while in Snowflake you are limited to insert only operations.

Lastly, Snowflake streams do not provide all the changes to a table. If a given row is updated multiple times in between batches, then only the most recent version of that row will be provided; therefore, it does not provide the true transactional history of a given dataset.

Snowflake Tasks

As stated by Snowflake, "Tasks can be combined with table streams for continuous ETL workflows to process recently changed table rows." In essence, tasks allow users to schedule code in the form of SQL statements, stored procedures, and procedural logic. This makes tasks a core component of any pipeline attempting to achieve streaming status on Snowflake.

Tasks can be scheduled to run at most every one minute. While running a job every minute sounds great, this latency limitation introduces severe

issues when processing data at scale. Add in the fact that Snowflake's limited support for advanced analytic workloads further reduces the viability of building intelligent production systems.

Table to Table Streaming

As discussed in the Spark Structured Streaming section, tables within Databricks act as streaming sources and sinks. This is a critical functionality, especially since Snowflake **requires** persisting data as a table in order to process it. For example, Databricks can transform data using the DataFrame API to read from and write to external systems (e.g. Kafka topic as a source and application database as a sink) without having to write the data to an intermediate table. This ability is absent in Snowflake, which is another example of Snowflake passing the complexity and unnecessary compute requirements onto the user. This is even the case when using Snowpark since it is simply different syntax to generate SQL commands. If you try to do stream processing within Snowflake you will need to write the data multiple times and read the data multiple times, drastically increasing processing time.

Below is a diagram showing how table to table streaming in Snowflake works. The intermediate objects are difficult to manage and the functionality does not truly provide streaming support. Snowflake is actively working to address this issue and has stated that a new solution is required in the form of dynamic tables, which is a private preview feature and not production ready.

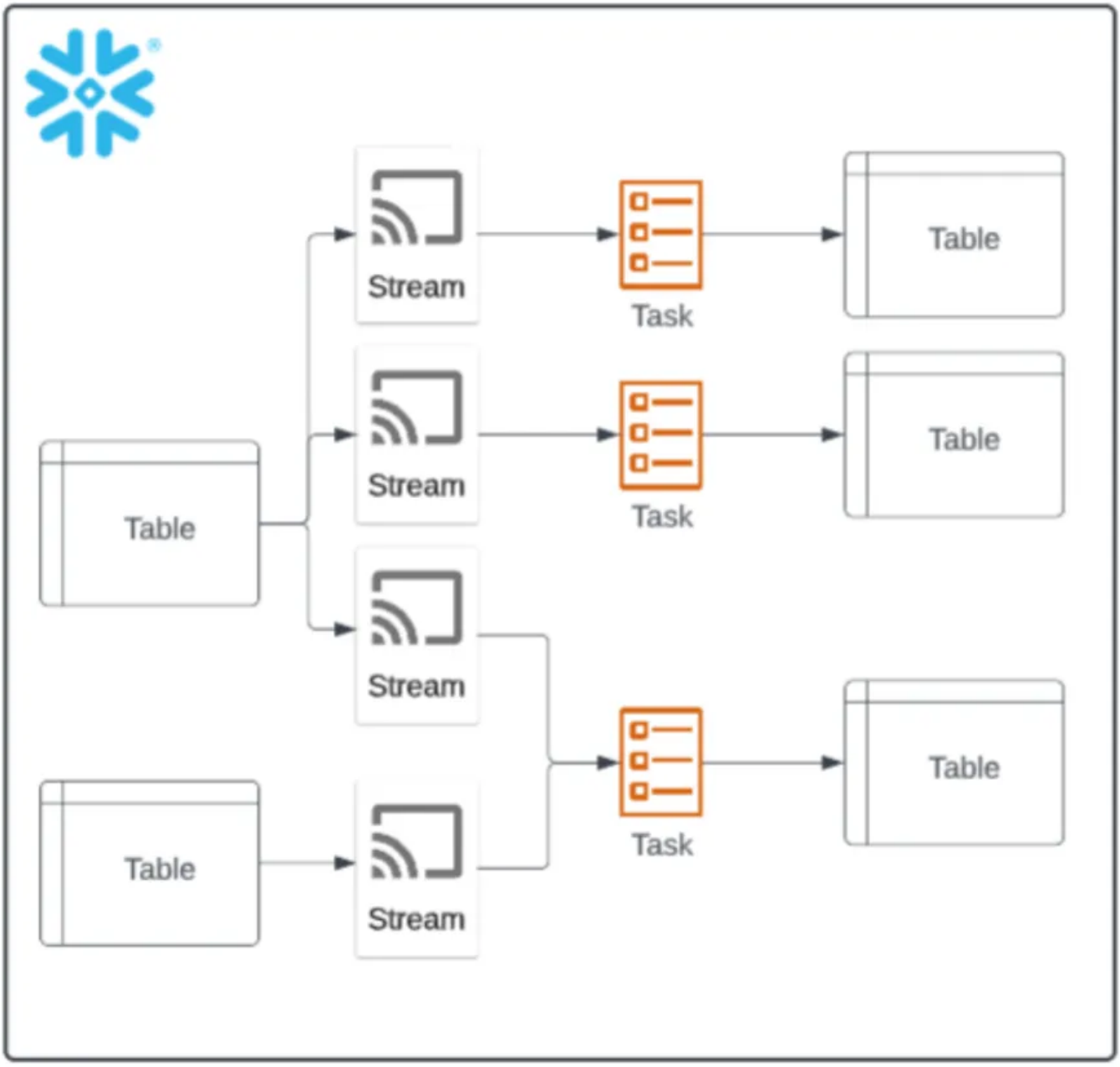


Table to Tale Streaming in Snowflake

To show the complexity of table to table streaming in Snowflake reference the code below, and assume that we have a table `events` with two columns `RECORD_CONTENT` and `RECORD_METADATA`.

- Create a Snowflake Stream for the `events` table called `events_stream`
- Create a Snowflake Task to parse our events table and create an append only sink table called `main_events`

- Create a Snowflake Stream to aggregate event data called

  `events_stream_agg`

- Create a Snowflake Task to write aggregate events to a table called

  `main_events_agg`

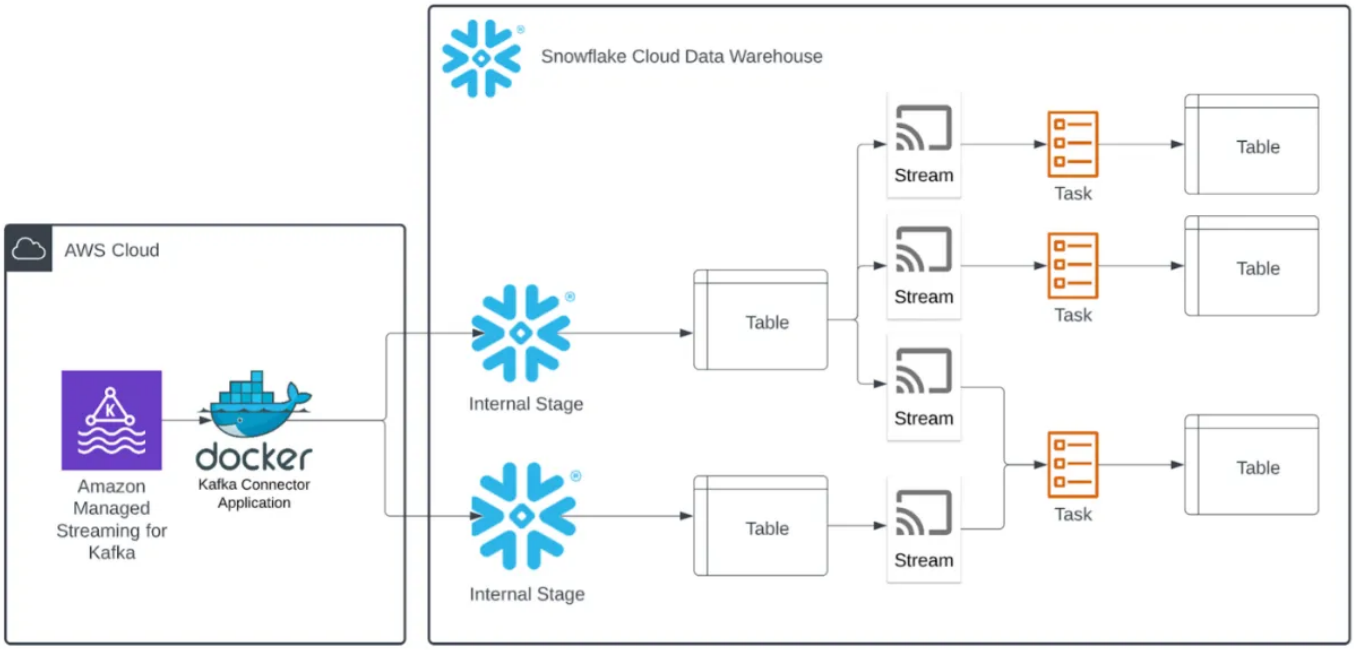
```
-- create stream object
create or replace stream events_stream
on table events
append_only=false
show_initial_rows=true;

-- task to write to table
create or replace task format_events
schedule = '1 MINUTE'
allow_overlapping_execution=false
user_task_managed_initial_warehouse_size = 'small'
as

insert into main_events

select record_metadata:eventId, record_metadata:action, record_metadata:time

from events ;

-- create stream object
create or replace stream events_stream_agg
on table events
append_only=false
show_initial_rows=true;

-- task to write to table
create or replace task agg_events
schedule = '5 MINUTE'
allow_overlapping_execution=false
user_task_managed_initial_warehouse_size = 'small'
as

insert into main_events_agg

select count(eventId) eventCount, action, time_slice(to_timestamp(time), 5, 'MIN
```

```
from main_events
group action, endTime;
```

The steps above are a lot more complex than what is required in Spark Structured Streaming. Further, if you require the ability to write to two different tables then you are limited to insert only operations or using temporary tables to write the data multiple times which requires unneeded overhead. More complex operations often require multiple reads, streams, and tasks to achieve the goal. This example shows when aggregating JSON data in Snowflake by timestamp column you need to first write the data to an intermediate table (`main_events`) then apply your aggregation. This results in three total tables in this pipeline.

Reading from a source table a single time and write to many tables is a critical ability for scalable streaming data pipelines. In our example where we are streaming events into a Snowflake table with two columns `RECORD_CONTENT` and `RECORD_METADATA`. As shown above, we can create a process that simply structures the data to make it available as a tabular dataset. Now if we want to aggregate this data every five minutes by event time we would need to create an entirely new process to do so, and have no ability to handle late arriving data without recomputing the entire window. Databricks elegantly handles late arriving data as described in the Spark Structured Streaming documentation.

Please see below for an end to end architecture diagram to build a data pipeline solution on Snowflake using change data capture. Note that depending on the solution size, it may be required to have multiple Kafka connectors. There is a much larger amount of complexity required when compared to the Databricks solution.



End to End Snowflake Streaming

Data Serving

Streaming solutions do not operate in isolation. Pipelines are meant to enable teams to orchestrate data movement between systems, implement real-time predictive analytics, and enable an organization to have proactive responses to events. The ability to integrate with the broader IT ecosystem is important to avoid bottlenecks in your processes.

Snowflake's primary data source is cloud object stores. When loading and unloading data with Snowflake engineers will need to stage the data in cloud storage prior to an external system being able to read that data. This will add additional data latency and application deployments to integrate Snowflake with the rest of your IT infrastructure. Snowflake does have a large number of connectors which allow customers to build custom applications that can query data via a Snowflake warehouse. The only way to bypass Snowflake compute to access data is when data is stored as external tables which have many limitations when compared to native Snowflake tables.

Snowflake's expectation is that customers need to integrate their business systems with Snowflake rather than Snowflake integrating with their systems. This is greatly contrasted to Databricks approach, which enables users to publish the data directly to external systems without requiring excess steps or applications.

Databricks has a robust set of native data sources that allow engineers to process data and write that data to an external system, such as an application database or a Kafka sink. With this capability data can be made available in external systems as soon as it is available within Databricks. Additionally, Databricks has a set of connectors that allow third party applications to connect to Databricks compute, which includes integrations with the most popular BI tools. Lastly, organizations have the ability to bypass Databricks entirely using open source tools that can connect directly to tables that are stored in the customer's cloud storage.

Conclusion

To simulate micro-batch streaming on Snowflake, users will need to use Streams (CDC functionality) and Tasks (scheduling), however, tasks cannot be scheduled to run less than one minute apart. This is a hard limitation not an option!

With Databricks, data is available in near real-time. Spark Structured Streaming natively runs on Databricks compute simplifying the architecture and developer experience, while integrating with delta tables for streaming sources and sinks. Databricks is the best engine for unified batch and stream data processing at scale.

Snowflake is a serviceable cloud data warehouse for historical BI analytics and reporting. Once data is in Snowflake the only reasonable option is to

batch process data. The attempt for real-time data in Snowflake quickly falls over and the cloud data warehouse becomes the biggest bottleneck in any streaming architecture.

For a product that advertises itself as “simple”, the streaming capabilities are far from it. Avoid streaming data and advanced analytic solutions on Snowflake. I encourage you to try out the various data pipeline solutions on Databricks, specifically the simple and managed pipelines of delta live tables.

Edit — May 17, 2023 — Updated overview table related to Databricks performance with the release of new performance benchmarks. Please see related blog.

For source code please reference the GitHub Repository.

Disclaimer: these are my own thoughts and opinions and not a reflection of my employer

Databricks Snowflake Apache Spark Streaming Streaming Data

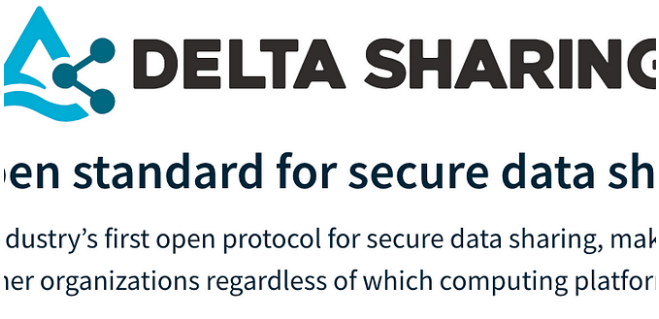
Written by Ryan Chynoweth

Edit profile

312 Followers

Senior Solutions Architect Databricks — anything shared is my own thoughts and opinions

More from Ryan Chynoweth



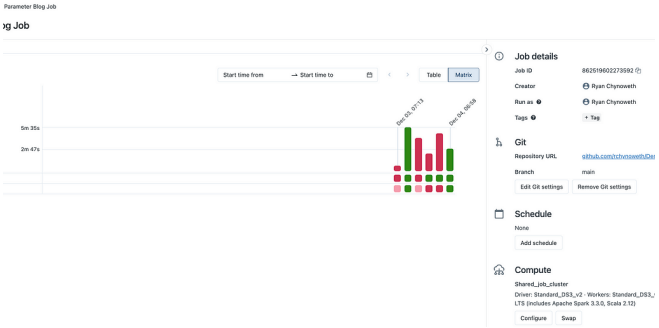
Ryan Chynoweth

Delta Sharing: An Implementation Guide for Multi-Cloud Architecture

Introduction

8 min read · 3 days ago

3 2



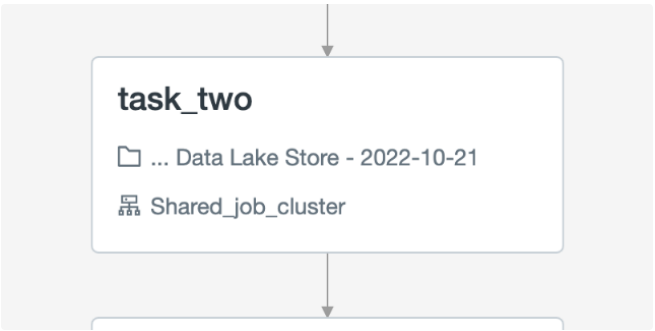
Ryan Chynoweth

Task Parameters and Values in Databricks Workflows

Databricks provides a set of powerful and dynamic orchestration capabilities that are...

11 min read · Dec 7, 2022

50 3



Ryan Chynoweth

Converting Stored Procedures to Databricks

Special thanks to co-author Kyle Hale, Sr. Specialist Solutions Architect at Databricks.

14 min read · Dec 29, 2022

116 5

Ryan Chynoweth

Recursive CTE on Databricks

Introduction

3 min read · Apr 20, 2022

33

See all from Ryan Chynoweth

Recommended from Medium

Karim Faiz

Mastering DBT: From Zero to Hero

Introduction

✨ · 12 min read · Dec 26, 2023

2

Daan Rademaker

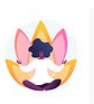
Do-it-yourself, building your own Databricks Docker Container

In my previous LinkedIn article, I aimed to persuade you of the numerous advantages o...

7 min read · Oct 16, 2023

26

Lists



Stories to Help You Grow as a Software Developer

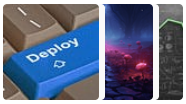
19 stories · 678 saves

data science and AI

38 stories · 31 saves

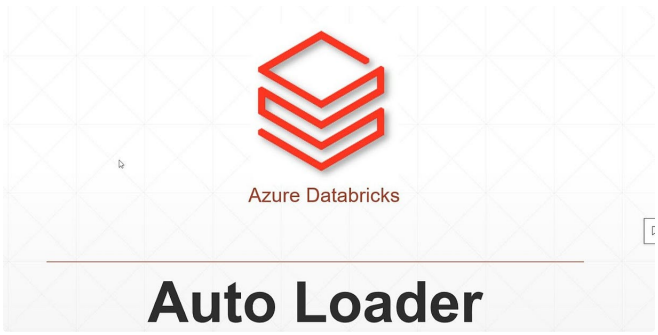
General Coding Knowledge

20 stories · 741 saves



Predictive Modeling w/ Python

20 stories · 749 saves



SIRIGIRI HARI KRISHNA in Towards Dev

Auto Loader

Autoloader simplifies reading various data file types from popular cloud locations like...

5 min read · Dec 9, 2023

4 1

Manasreddy

How to pass: Databricks Data Engineer Professional Certification

Conquering the Databricks Data Engineer Professional Exam: A Definitive Guide

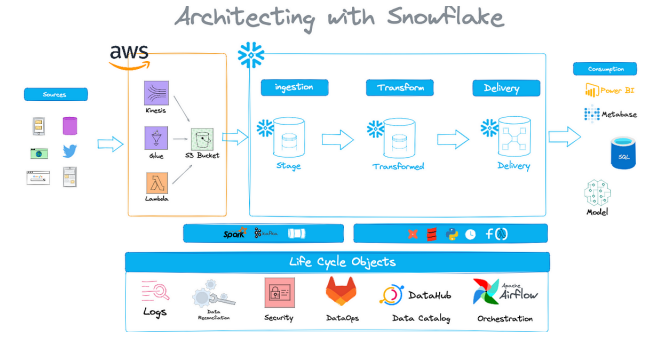
3 min read · Aug 24, 2023

8

Tahar Chanane

The Future is Meta-Data Driven: Pioneering a New Digital...

In the corridors of the digital realm, we often find ourselves inundated with data. Massive...



Cássio Bolba in AI Mind

Snowflake Ingestion Methods

Snowflake data platform allows data engineers to ingest data in many ways. Let's...

6 min read · Aug 28, 2023

101

✨ · 5 min read · Dec 21, 2023

951 16

See more recommendations