Streaming CDC data from PostgreSQL to Snowflake challenges and solutions

Alexandru Cristu
Senior Solution Architect

Alexandru Cristu

- Senior Solution Architect @ Streamkap
- 15+ years background in building high throughput, low latency, geographically distributed custom software solutions
- Passionate about data streaming and stream processing, kappa architecture, event sourcing, CQRS

Agenda

- Overview, architecture and setup
- Backfill, initial/incremental/blocking snapshots, pros/cons
- Postgres challenges: wal log growth, TOAST
- Snowflake challenges: most recent view, channel offsets, costs
- Pipeline challenges: complete and correct mapping, interruptions, monitoring, streaming transforms

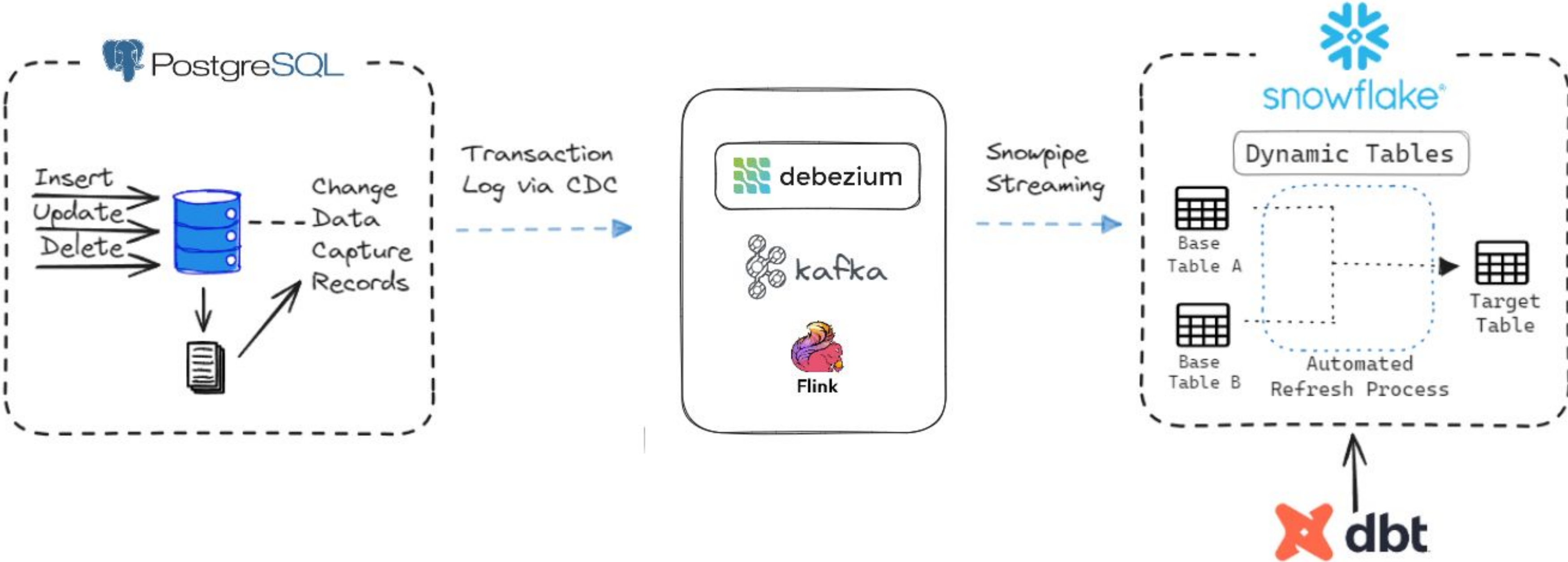
Overview

- CDC - Change Data Capture refers to the process of capturing changes made to data in a source system such as a database
- Streaming ETL (extract, transform, load) is a type of data integration process that involves continuously extracting data from various sources, transforming it to fit the needs of the destination system, and loading it into the destination system in near real-time.

Why CDC for Streaming ETL ?

- Much lower end-to-end latency - power real-time customer facing analytics, operations, personalization
- Lower costs
- Minimal load on source database
- Retain historical data for auditing, time travel, etc

Architecture



streamkap

Setup PostgreSQL - privileges

```
CREATE USER streamkap_user PASSWORD '{password}';

-- Create a role for Streamkap

CREATE ROLE streamkap_role;

GRANT streamkap_role TO streamkap_user;

-- Grant Streamkap permissions on the database, schema and all tables to capture

GRANT CONNECT ON DATABASE "{database}" TO streamkap_role;

GRANT CREATE, USAGE ON SCHEMA "{schema}" TO streamkap_role;

GRANT SELECT ON ALL TABLES IN SCHEMA "{schema}" TO streamkap_role;

ALTER DEFAULT PRIVILEGES IN SCHEMA "{schema}" GRANT SELECT ON TABLES TO streamkap_role;
```

Setup PostgreSQL - logical replication

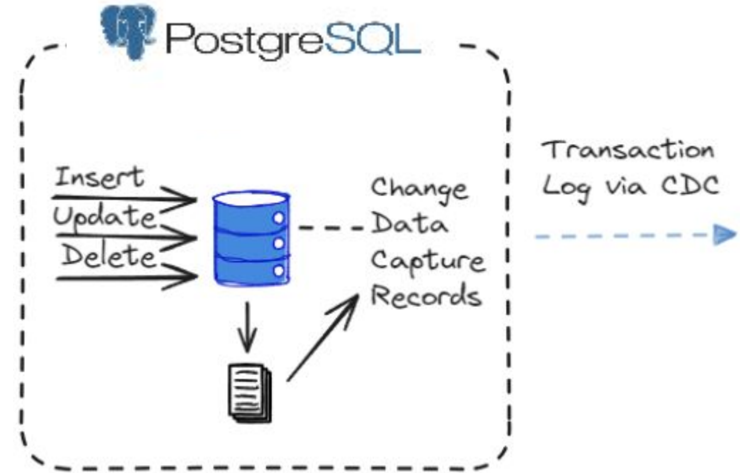
```
-- Option 1: All Tables
CREATE PUBLICATION streamkap_pub FOR ALL TABLES;

-- Option 2: Specific Tables
CREATE PUBLICATION streamkap_pub FOR TABLE table1, table2, table3, ...;

-- PostgreSQL 13 or later, enable the adding of partitioned tables
-- CREATE PUBLICATION streamkap_pub FOR ALL TABLES WITH (publish_via_partition_root=true);

-- Create a logical replication slot
SELECT pg_create_logical_replication_slot('streamkap_pgoutput_slot', 'pgoutput');

-- Verify the table(s) to replicate were added to the publication
SELECT * FROM pg_publication_tables;
```


streamkap

Real-Time Ingestion

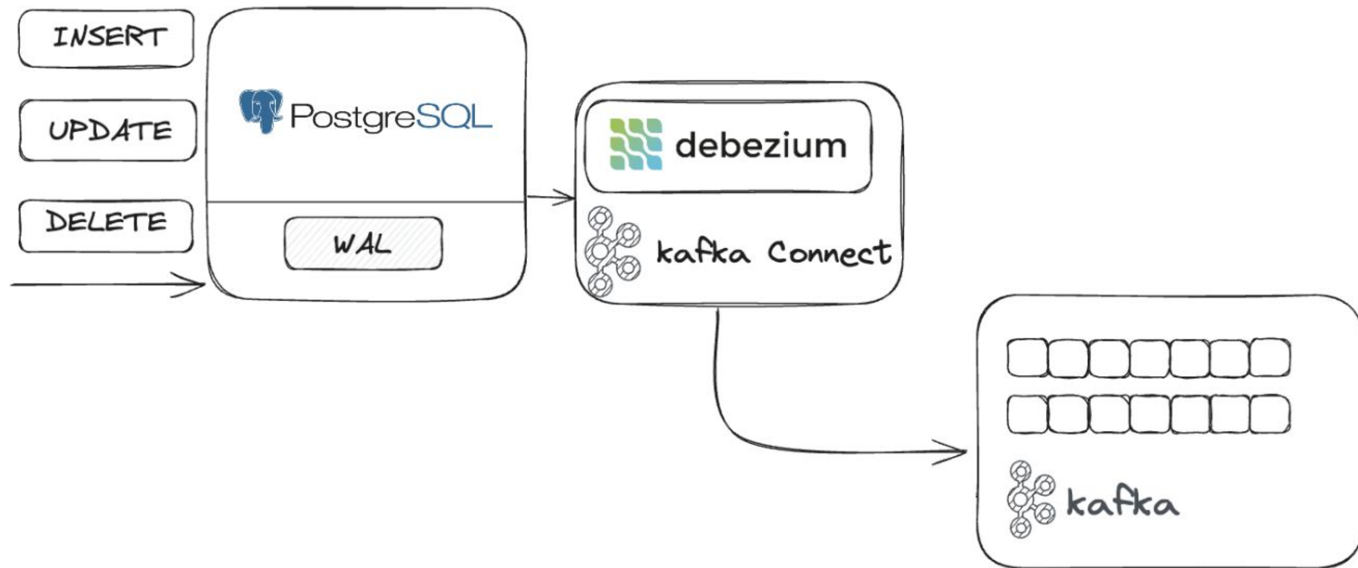
Debezium is the the de-facto way to stream CDC data and replicate the database

- All changes in the database are written to the database log (WAL)
- Debezium connectors read the log in real-time



streamkap

Real-Time Ingestion



streamkap

Debezium Connectors

- Wide support for all major databases
- Kafka Connect compatible
- Data can be snapshotted/backfilled before moving on to streaming mode
- Practically zero impact on the database (Streaming mode)
- Community support is excellent

Setup Snowflake - warehouse, user

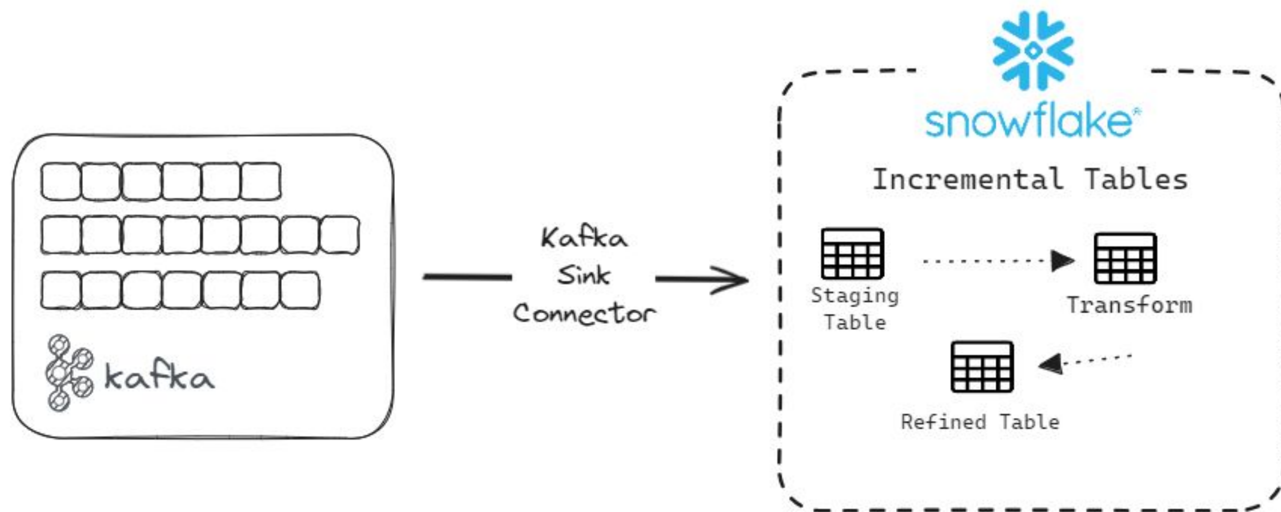
```
SET user_name           = UPPER('STREAMKAP_USER');
SET user_password       = '{password}'; -- IMPORTANT: Make sure to change this!
SET warehouse_name      = UPPER('STREAMKAP_WH'); -- Used for optional views not ingestion
SET database_name       = UPPER('STREAMKAPDB');
SET schema_name         = UPPER('STREAMKAP');
SET role_name           = UPPER('STREAMKAP_ROLE');
SET network_policy_name = UPPER('STREAMKAP_NETWORK_ACCESS');

. . .

-- Create a Snowflake role with privileges for the Streamkap connector
USE ROLE IDENTIFIER($securityadmin_role);
CREATE ROLE IF NOT EXISTS IDENTIFIER($role_name);
GRANT . . . . .

-- ALTER NETWORK POLICY STREAMKAP_NETWORK_ACCESS SET ALLOWED_IP_LIST=('x.y.z.t');
CREATE NETWORK POLICY IDENTIFIER($network_policy_name) ALLOWED_IP_LIST=('x.y.z.t');
ALTER USER IDENTIFIER($user_name) SET NETWORK_POLICY = $network_policy_name;
```

Setup Snowflake - key authentication

```
# generates an encrypted RSA private key
openssl genrsa 2048 | openssl pkcs8 -topk8 -v2 aes256 -inform PEM -out streamkap_key.p8
-passout pass:{passphrase}

# generates the public key, referencing the private key
openssl rsa -in streamkap_key.p8 -pubout -out streamkap_key.pub -passin pass:{passphrase}

egrep -v '^-|^$' ./streamkap_key.pub | pbcopy

# Update snowflake user
SET user_name = UPPER('STREAMKAP_USER');

USE ROLE SECURITYADMIN;

-- Replace '{public key}' below with the public key file contents
-- If you used the previous command to copy the key to your clipboard, use Ctrl+V (Windows)
-- or Cmd+V (MacOS) to replace the '{public key}' placeholder with the key
-- Key part MUST start with 'MII' excluding any headers and footers
ALTER USER STREAMKAP_USER SET RSA_PUBLIC_KEY = '{public key}';
```

Snowpipe Streaming

- low-latency loading of streaming data rows
- Use the Snowpipe Streaming API in streaming scenarios where data is streamed via rows (for example, Apache Kafka topics) instead of written to files

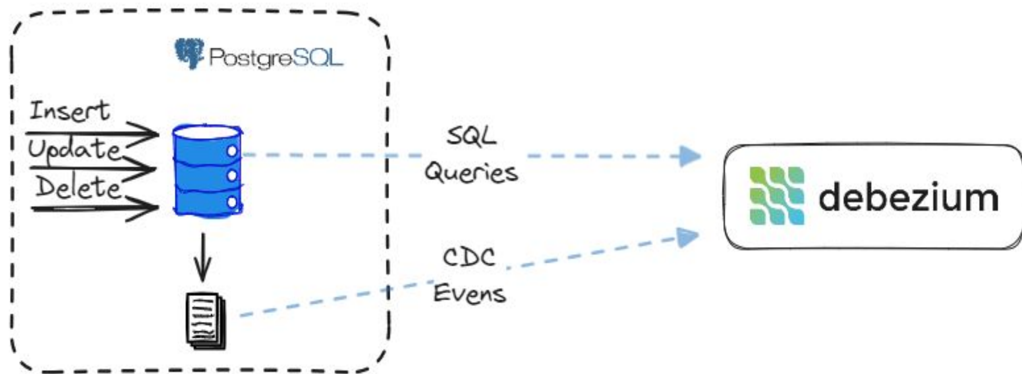


streamkap

Backfill

Snapshotting refers to the process of loading existing data from PostgreSQL into Snowflake.

- Initial Snapshots
- Incremental Snapshots
- Blocking Snapshots



streamkap

Backfill - Initial Snapshots

A blocking snapshot run before streaming is started.

Pros:

- These can be run concurrently and are very fast
- Captures the entire current state of the database tables

Cons:

- High impact on system resources, especially for large tables and when using higher parallelism
- Streaming is paused, for large databases WAL growth can become an issue
- Connector restart means complete re-snapshot from scratch

streamkap

Backfill - Incremental Snapshots

A chunked snapshot running in parallel with streaming.

Pros:

- Not causing any WAL growth
- Low impact on the source database

Cons:

- Slower than blocking snapshots
- Currently can miss TOAST-ed values

streamkap

Backfill - Blocking Snapshots

Blocking snapshots, very similar to initial snapshots, but can be triggered anytime.

Pros:

- Can be triggered for specific tables
- Can use additional conditions to filter records

Cons:

- Same cons as initial snapshots
- connector restart can cause a full initial snapshot

What are Dynamic Tables?

Declarative transformation

Automated data refresh

Built for data pipelines

Dynamic tables simplify data engineering by providing automated and reliable data transformation pipelines in Snowflake.

Snowflake - most recent view

Snowpipe streaming is append only, the complete changes history is ingested into the snowflake target tables.

Querying the latest view of the data requires de-duplicating the changes history with:

- Dynamic tables
- Snowflake connector upsert mode

How to create Dynamic Table?

```
{{
  config(
    materialized = 'dynamic_table',
    target_lag = '5 minutes',
    snowflake_warehouse = 'my_warehouse'
  )
}}

SELECT *
FROM my_db.my_schema.my_table
WHERE my_value IS NOT NULL
```

Convert streamkap CDC events table to Most Recent State table using Dynamic Table

```
CREATE OR REPLACE DYNAMIC TABLE my_db.my_schema.my_table
TARGET_LAG = '5 minutes'
WAREHOUSE = dynamic_tables_wh
AS (
    WITH cdc_events AS (
            SELECT * FROM my_db.streamkap_schema.original_table
        ),

        most_recent_record_state_table AS (
            SELECT *
                EXCLUDE (dedupe_id, __deleted),
                    TO_BOOLEAN(__deleted) as __deleted,
                    TO_TIMESTAMP(_streamkap_ts_ms, 3) as _streamkap_ts
            FROM (
                SELECT
                    *,
                    ROW_NUMBER() OVER (
                        PARTITION BY
                            id
                        ORDER BY
                            CREATED_ON DESC,
                            _streamkap_ts_ms DESC
                    ) AS dedupe_id
                FROM
                    cdc_events
            )
            WHERE
                dedupe_id = 1
            AND __deleted = 'false')

        SELECT *
        FROM most_recent_record_state_table
        )
    ;
```

Snowpipe Streaming Costs

Snowpipe Streaming does not need a warehouse to run keeping costs down

- Snowpipe Streaming has two charges: Cloud service and migration.
- Example monthly cost running 24/7, $2 credit cost, 1 TB
  - Cloud Service: 30 days * 24 hrs * 1 client * 0.01 credits = 7.2 credits x $2 = $14.4
  - Migration: automated background process, approximately 3-10 credits per TB of data (Assuming 6 credits per TB) = 6 credits * $2 = $12
  - Total = $26.4/TB
- **Takeway: $20-$34/TB!**

Real World Example - Snowpipe Streaming + Dynamic Tables

“Snowflake costs for pipeline from our batch ETL tool is around 190 credits / per week. Compared to Snowpipe Streaming usage from Streamkap which 1-2 credits per week.

In terms of modeling, around 200 credits / per week before and now it’s around 130 credits / per week with Dynamic Tables”

Before: 390 credits/week - **$3,354/month** ($2/credit), 20 min latency

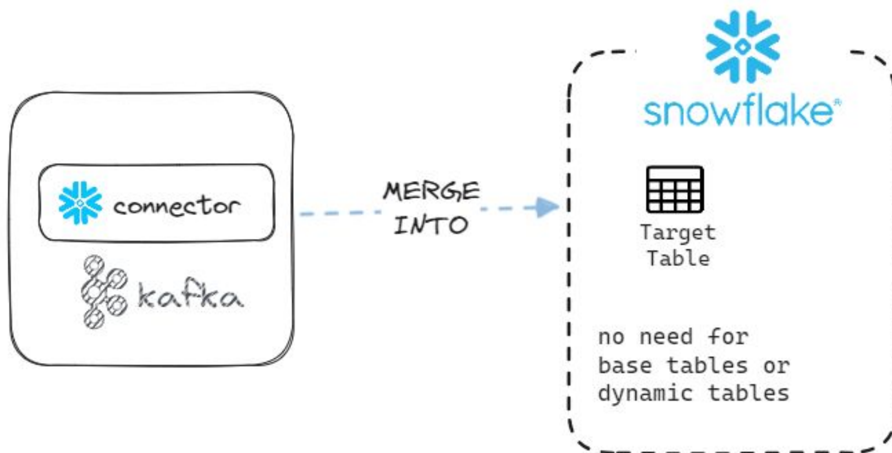
After: 132 credits/week - **$1,135/month** ($2/credit), 5 mins latency

4x faster and 3x cheaper

Snowflake - upsert

Connector sends bulk MERGE INTO statements to ensure the target tables contain the latest view of the data.

Costs depend on the size of the warehouse, billing will run continuously, warehouse will not be able to auto-suspend.



streamkap

Snowflake - challenge with offsets

Snowpipe streaming channels maintain kafka offsets, this is part of offering exactly once guarantees.

Replaying event with past offsets has no effect and this comes with challenges when running maintenance work like data conversions and kafka migrations.

Streaming CDC data, in most use case, works well with at least once guarantees, which means in certain error scenarios some events can be replayed leading to duplicates. However deduplicating the latest view of the data with dynamic tables or MERGE INTO statement is needed anyway.

Challenge - type mapping completeness

Kafka Connect/PostgreSQL Data Type	Snowflake Data type
INT8/16/32/64	BYTEINT/SMALLINT/INT/BIGINT
FLOAT32/64	FLOAT/DOUBLE
BOOLEAN	BOOLEAN
BYTES	BINARY
STRING	VARCHAR

Type mapping/conversions

Kafka Connect/PostgreSQL Data Type	Snowflake Data type
`org.apache.kafka.connect.data.Decimal`	DECIMAL

`org.apache.kafka.connect.data.Timestamp`
`io.debezium.time.ZonedTimestamp`
 	
`io.debezium.time.MicroTimestamp`
	TIMESTAMP

`org.apache.kafka.connect.data.Date`	DATE

`io.debezium.time.Time`
 	
`io.debezium.time.MicroTime`
	TIME(3), TIME(6)

Type mapping/conversions

Kafka Connect/Debezium/PostgreSQL Data Type	Snowflake Data type
`io.debezium.data.Json (json, hstore)`	VARIANT
STRUCT	VARIANT
ARRAY	VARIANT

`io.debezium.data.geometry.Geometry`
`io.debezium.data.geometry.Geography`
`io.debezium.data.geometry.Point`
	VARIANT

Challenge - snowflake unavailable

We had situation where destination was unavailable for several hours up to 2 days (403 errors).

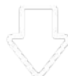
If destination is down source must keep consuming to prevent WAL size issues.

In most use cases Kafka can easily ingest and store events for 2 days or more. Snowflake sink connector high throughput of 100k records/sec or more, depending on the number of tables/channels.

Challenge - monitoring dev/staging noise

Monitoring large number of pipelines can be challenging, important metrics:

- Source throughput and ms lag behind source database
- Destination throughput and event lag
- Kafka throughput, topics size

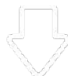


Alerts/Dashboards affected by dev/staging pipelines, especially in a larger multi-tenant environment

Challenge - pinpoint issues

Drilling down to table/record/field level is time consuming with many source databases and tables.

We've seen use cases with 500+ tables in one database and cases where there are 100+ databases with tens of tables each, custom monitoring required:



- estimated/exact counts per table in source vs destination
- Periodic sampling of records from source vs destination with consistency checking for each field

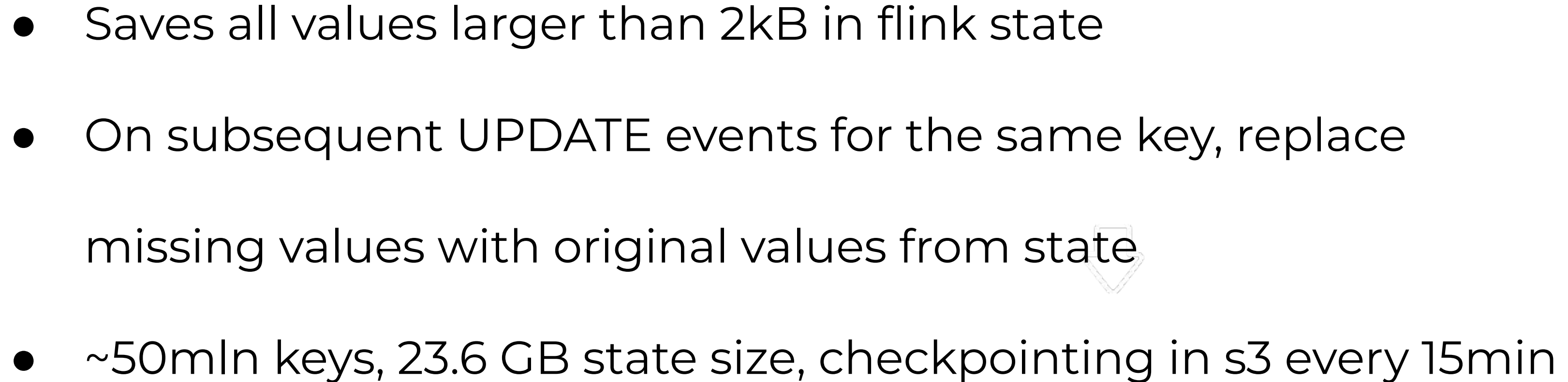
Challenge - TOAST handling

Update events will not receive TOAST-ed values unless they are modified, solutions:

- `REPLICA IDENTITY FULL` - it will create a huge WAL size
- Debezium reselect post processor, record by record, not practical for large write throughput where TOAST values don't change often.
- Stream processing - keep original values in cache and replace missing values in future update events

Stream processing - TOAST

Apache flink DataSteam API generic job:



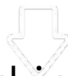
- Saves all values larger than 2kB in flink state
- On subsequent UPDATE events for the same key, replace missing values with original values from state
- ~50mln keys, 23.6 GB state size, checkpointing in s3 every 15min

streamkap

Stream processing - low latency transforms

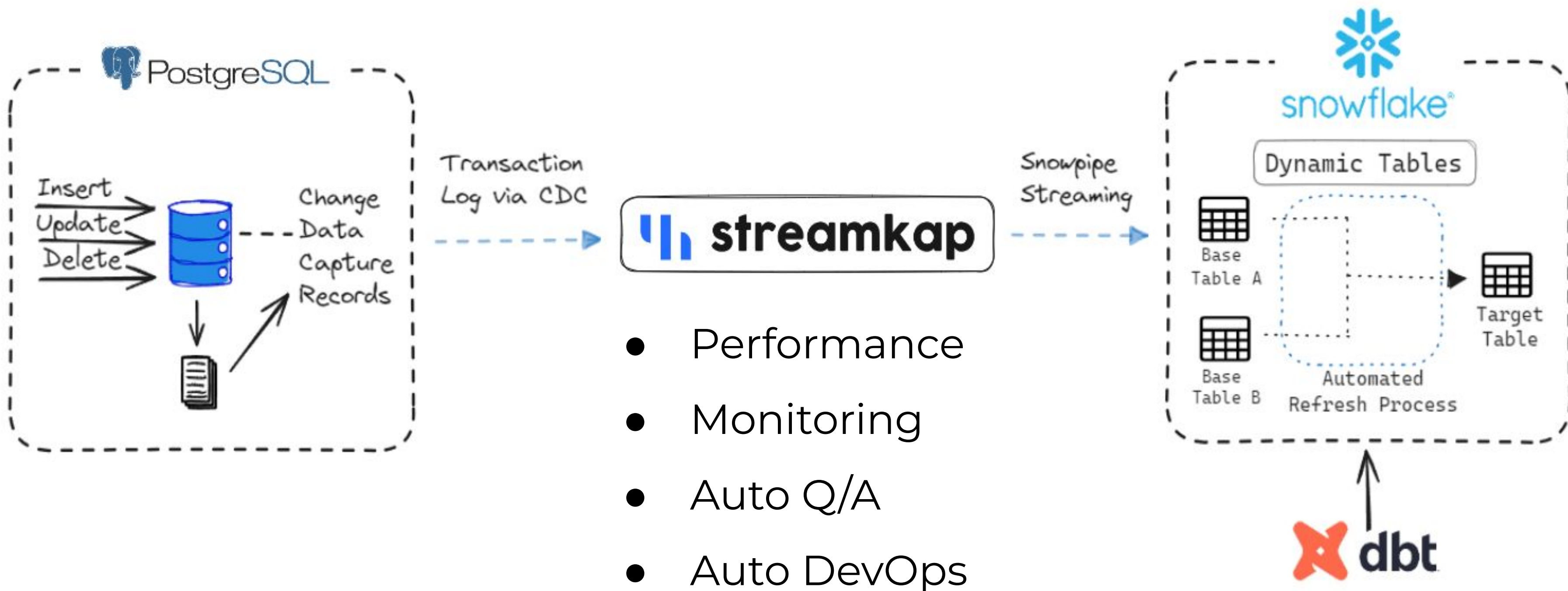
Apache flink layer can be used to apply transformations before ingestion into snowflake:

- Data conversions, fix inconsistencies in semi-structured data
- Array unrolling, generating new record for each nested array element
- Enrichment, windowed and non-windowed joins



If ingesting the same data in multiple destinations, applying transforms in the steaming layer would be more cost effective.

Architecture - using Streamkap



- Performance
- Monitoring
- Auto Q/A
- Auto DevOps

streamkap

Thank you

Useful links

- PostgreSQL - postgresql.org
- Snowflake - docs.snowflake.com
- Debezium - debezium.io
- Kafka - kafka.apache.org
- Flink - flink.apache.org
- Streamkap - streamkap.com

Alexandru Cristu
Senior Solution Architect