REAL-TIME DATA ENGINEERING

PostgreSQL → ClickHouse CDC

Production-ready SQL, YAML, and operational patterns for streaming change events with Estuary Flow — without standing up a Kafka cluster.

STACK
Postgres 15+ · ClickHouse · Estuary Flow

FORMAT
Copy-paste snippets · 8 pages

USE CASE
Sub-second OLTP → OLAP replication

AUDIENCE
Data & Platform Engineers

COMPANION TO

"Real-Time CDC from PostgreSQL to ClickHouse with Estuary Flow" — the 12-minute hands-on walkthrough covering architecture, schema evolution, fan-out, and production hardening.

v1.0

JULY 2026 · Vertex Frontier

01 · ARCHITECTURE

The Three-Stage Pipeline

Estuary Flow decouples the source from destinations through a durable middle layer. The collection is **not** a transient queue — it's a stored dataset on cloud object storage. Every destination reads from the same captured data independently, so adding a new sink never touches Postgres again.

KAFKA ENGINEERS

This maps 1:1 to `Source Connector → Kafka Topic → Sink Connector`. Estuary did not invent decoupled streaming — the architectural distinction is that collections are indexed journals on cloud object storage (Gazette), not broker-local disk retention.

Exactly-once delivery is engineered into the Gazette consumer framework itself — not something you configure around with idempotency keys. The watermarks table in Postgres is the bridge that lets the backfill process know precisely where history ends and the live tail begins.

02 · SOURCE SETUP

PostgreSQL Configuration — 4 Steps

The entire Postgres side is ~10 lines of SQL. Get these right and you avoid a week of silent-failure debugging later. Step 3 is the one most teams miss.

Step 1 — Enable logical replication

SQL — POSTGRESQL.CONF (THEN RESTART)

```
-- Self-hosted: set in postgresql.conf, then restartALTER SYSTEM SET wal_level = logical;

-- Cloud (RDS, Cloud SQL, Neon, Supabase):-- set via console / parameter group instead
```

NEON WARNING

Enabling logical replication changes `wal_level` for **every database in the project**, cannot be reverted, and restarts all computes — dropping active connections.

Step 2 — Create the capture user

SQL — DEDICATED REPLICATION USER

```sql
CREATE USER flow_capture WITH PASSWORD'secret'REPLICATION;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO flow_capture;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO flow_capture;

-- Required for schema auto-discovery on Postgres 14+:GRANT SELECT ON ALL TABLES IN SCHEMA
information_schema, pg_catalog TO flow_capture;
```

Step 3 — Watermarks table (do NOT skip)

SQL — WATERMARKS TABLE + PUBLICATION

```sql
CREATE TABLE IF NOT EXISTS public.flow_watermarks (
    slot TEXT PRIMARY KEY,
    watermark TEXT
);
GRANT ALL PRIVILEGES ON TABLE public.flow_watermarks TO flow_capture;

CREATE PUBLICATION flow_publication;
ALTER PUBLICATION flow_publication ADD TABLE
    public.flow_watermarks,
    customers,
    orders,
    products;
```

THE SILENT STALL

If `flow_watermarks` is left out of the publication, the capture connection still succeeds — and the backfill hangs forever. No error message. If a backfill never finishes, check the publication list **first**.

Step 4 — Replica identity + WAL cap

SQL — FULL ROW IMAGES + WAL SAFETY NET

```sql
-- Log the complete row before & after updatesALTER TABLE customers REPLICA IDENTITY FULL;
ALTER TABLE orders    REPLICA IDENTITY FULL;
ALTER TABLE products  REPLICA IDENTITY FULL;

-- Prevent WAL bloat from taking down Postgres-- if the CDC consumer loses connectivityALTER
SYSTEM SET max_slot_wal_keep_size = '50GB';
SELECT pg_reload_conf();
```

WHY REPLICA IDENTITY FULL?

Without it, TOASTed values (large text/jsonb) that are unchanged during an update are omitted from the WAL entirely — showing up downstream as nulls. The cost is a heavier WAL, which is why you pair it with `max_slot_wal_keep_size`.

Estuary Flow YAML Templates

Both capture and materialization can be defined via dashboard or committed YAML. The YAML form is what you check into Git and deploy via `flowctl` for CI/CD.

Capture — Postgres source

YAML — CAPTURES/ESTUARY.YAML

```yaml
captures:
  acme/commerce-capture:
    endpoint:
      connector:
        image: ghcr.io/estuary/source-postgres:v3
        config:
          address: host:port
          database: postgres
          user: flow_capture
          credentials:
            auth_type: UserPassword
            password: ${POSTGRES_PASSWORD}bindings:
      - resource: { stream: customers, namespace: public }
        target: acme/customers
      - resource: { stream: orders,    namespace: public }
        target: acme/orders
      - resource: { stream: products,  namespace: public }
        target: acme/products
```

Materialization — ClickHouse sink

YAML — MATERIALIZATIONS/CLICKHOUSE.YAML

```yaml
materializations:
  acme/clickhouse-materialization:
    endpoint:
      connector:
        image: ghcr.io/estuary/materialize-clickhouse:v1
        config:
          address: your-clickhouse-host:9440
          database: my_database
          credentials:
            auth_type: user_password
            username: flow_user
            password: ${CH_PASSWORD}bindings:
      - { resource: { table: customers }, source: acme/customers }
      - { resource: { table: orders    }, source: acme/orders }
      - { resource: { table: products  }, source: acme/products }
```

Bonus — OpenTelemetry collector

YAML — OTEL-COLLECTOR-CONFIG.YAML

```yaml
# Replace {org_id} with your Estuary organization ID.# Set ESTUARY_API_TOKEN and DD_API_KEY as
env vars.receivers:
  prometheus:
    config:
      scrape_configs:
        - job_name: 'estuary_flow_metrics'scrape_interval: 10s
          metrics_path: '/api/v1/organizations/{org_id}/metrics'bearer_token:
'${ESTUARY_API_TOKEN}'scheme: https
          static_configs:
            - targets: ['api.estuary.dev']
exporters:
  datadog:
    api:
      key: '${DD_API_KEY}'service:
  pipelines:
    metrics:
      receivers: [prometheus]
      exporters: [datadog]
```

04 · QUERY PATTERNS

ClickHouse: ReplacingMergeTree & FINAL

Updated rows arrive as **new inserts**. ClickHouse deduplicates them in the background using `flow_published_at` as the version column — but background merges are eventual, not immediate. A plain `SELECT` can return stale or duplicate rows.

SQL — CORRECT WAY TO QUERY A CDC-FED TABLE

```sql
SELECT order_id, status, discount_cents
FROM orders FINALWHERE order_id = 4;
```

FINAL IS A TOOL, NOT A DEFAULT

On a 500M-row table in a high-concurrency BI dashboard, FINAL will crush CPU. Reserve it for targeted point queries. For dashboards, build a Materialized View on top of the ReplacingMergeTree, or use `argMax()` aggregation.

Merge vs Delta updates

Setting	What lands in ClickHouse	When to use
Merge `(default)`	One current-state row per key. "pending → paid → shipped" stays a single row that updates in place.	Live mirror of source tables. Operational dashboards. Latest-state queries.
Delta	Every insert/update/delete is its own new row. Append-only change log. You collapse it at query time.	Audit trails. Compliance. "What happened" analytics. High-volume tables where state reduction isn't needed.

Soft vs Hard deletes

Mode	Behavior	When to use
Soft `(default)`	Row stays. A `_meta_op` column records `c` / `u` / `d`.	Compliance / audit. "This record used to exist" is itself valuable.
Hard	Tombstone row with `_is_deleted = 1` is inserted. ReplacingMergeTree eventually purges it.	Mirror production state one-to-one. A deleted order should actually disappear.

05 · PRODUCTION

Production Readiness Checklist

Run through this before promoting a CDC pipeline to production. Print it, share it with the on-call rotation, and re-run it after every infrastructure change.

- [ ] **WAL level set to logical**
  Verified on Postgres primary via `SHOW wal_level;`
- [ ] **Watermarks table in publication**
  `flow_watermarks` explicitly added via `ALTER PUBLICATION ... ADD TABLE`
- [ ] **WAL retention cap set**
  `max_slot_wal_keep_size = '50GB'` to prevent storage collapse if consumer goes offline
- [ ] **Replica identity full on TOASTed tables**
  Any table with large `text` / `jsonb` columns
- [ ] **ClickHouse deduplication strategy**
  Queries use `FINAL` for point lookups, MV or `argMax()` for dashboards
- [ ] **Replication lag monitoring**
  Prometheus / Datadog alert on `lag_bytes > 500MB` for 5+ minutes
- [ ] **Critical lag alert**
  Page on `lag_bytes > 10GB` or `active = false`
- [ ] **HA failover strategy**
  `pg_failover_slots` on self-hosted, or verified managed slot re-creation
- [ ] **Schema evolution review process**
  Source-side review paired with auto-propagation — not blind trust
- [ ] **Mass DML batching policy**
  Chunked in 10K–50K row batches with sleep intervals; no unbatched `UPDATE ... WHERE` on big tables

06 · TROUBLESHOOTING

Error → Cause → Fix Matrix

Log error	Root cause	Immediate fix
`replication slot "flow_slot" is active for PID 1234`	Previous connection process crashed without dropping its session.	`SELECT pg_terminate_backend(1234);`
`Backfill status stalled at 0%`	`flow_watermarks` omitted from `PUBLICATION`.	`ALTER PUBLICATION flow_publication ADD TABLE flow_watermarks;`
`DB::Exception: ReplacingMergeTree, query without FINAL`	App code queries dedup tables without forcing a merge context.	Append `FINAL`, or build a materialized view for dedup.
`Derivation Panic: TypeScript memory limit exceeded (OOM)`	Heavy in-memory aggregation or unoptimized regex in a derivation.	Keep derivations stateless. Push heavy joins to ClickHouse or dbt.
`Collection Schema Validation Failed (Type Mismatch)`	Source emitted a breaking type change not mapped in JSON schema.	Update collection binding schema, or set schema evolution to permissive.
`Materialization Throttle: Gazette Journal Backpressure`	ClickHouse cluster overwhelmed, can't keep up with batch flushes.	Increase ClickHouse block insert size, or scale ClickHouse compute.

LONG TRANSACTION TRAP

If a background worker holds an open transaction for 3 hours, Postgres holds the slot's `restart_lsn` at the *beginning* of that transaction. After a connector restart, you'll re-read hours of historical WAL. Estuary mitigates this by embedding heartbeats and progress watermarks into the replication log.

07 · MONITORING

Replication Lag & Alerting

Sub-second CDC latency is only reliable if you monitor slot lag proactively. Expose this view to Prometheus or Datadog.

SQL — QUERY ACTIVE REPLICATION LAG

```sql
SELECT
    slot_name,
    active,
    pg_size_pretty(
      pg_wal_lsn_diff(pg_current_wal_lsn(), confirmed_flush_lsn)
    ) AS bytes_behind,
    pg_wal_lsn_diff(pg_current_wal_lsn(), confirmed_flush_lsn) AS lag_bytes
FROM pg_replication_slots
WHERE slot_name = 'flow_slot';
```

Warning threshold

`lag_bytes > 500 MB` for > 5 min
→ Network throttling or consumer backpressure.

Critical threshold

`lag_bytes > 10 GB` or `active = false`
→ Approaching `max_slot_wal_keep_size` limit. Page on-call.

08 · PRO TIPS

Senior Engineer Notes

01 · DECOUPLING COMPOUNDS

The first destination feels like the main event. The third, fourth, and fifth destinations are where the architecture pays for itself — none of them touch Postgres again.

02 · FINAL IS A TOOL, NOT A DEFAULT

For high-traffic dashboards, a scheduled `INSERT ... SELECT ... FINAL` into a secondary table, refreshed on an interval, is usually the better trade than running FINAL on every query.

03 · SCHEMA EVOLUTION IS A GOVERNANCE DECISION

Automated propagation means a careless `ALTER TABLE` in prod flows downstream automatically. Pair auto-evolution with a source-side review process — not blind trust.

04 · ESTUARY IS NOT A STREAMING DATABASE

For complex real-time window aggregations and stateful multi-table JOINs *before* reaching ClickHouse, complement Estuary with RisingWave or Materialize via Dekaf.

05 · PLAN YOUR EXIT FROM DAY ONE

Dekaf exposes collections as Kafka topics — point Flink or Kafka Connect at it for zero-re-ingestion egress. TypeScript derivations map 1:1 to dbt (batch) or Flink SQL (streaming), but they will need a rewrite.

DIY cost formula

HONEST COMPARISON

DIY Monthly Cost = Infrastructure Spend + (Maintenance Hours × Hourly Rate)

The real value isn't precision — it's forcing the DIY cost comparison to include engineer time, which almost never gets counted honestly. Estuary's free tier covers 10 GB/month; beyond that, pricing scales by GB transferred through collections. Note: `REPLICA IDENTITY FULL` increases total captured bytes vs key-only deltas.

END OF CHEAT SHEET

Build it. Break it. Ship it.

The fastest way to evaluate this stack is the same one used throughout the companion article: stand up a small Postgres schema, capture it, materialize it into ClickHouse, then deliberately run an `ALTER TABLE` in production while it's live. How that moment plays out tells you everything.

REFERENCES

Estuary Flow docs · estuary.dev
PostgreSQL Logical Replication · postgresql.org
ClickHouse ReplacingMergeTree · clickhouse.com
Altinity ReplacingMergeTree deep dive · altinity.com
Prodege & Forward case studies · estuary.dev/customers

PUBLISHED BY VERTEX FRONTIER · V1.0 · JULY 2026
COMPANION TO "REAL-TIME CDC: POSTGRESQL → CLICKHOUSE WITH ESTUARY FLOW"