AMP

From batch processing to streaming

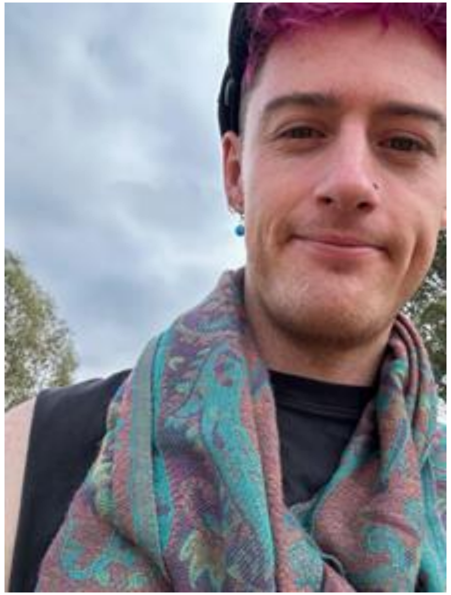
Transitioning Private Cloud ClickHouse OSS to AWS Cloud Native

Intro

Software engineer for 7 years, recently started into the world of Data Engineering as our product grew more complex

Ran startup agency, brought over 30 new SaaS products to market

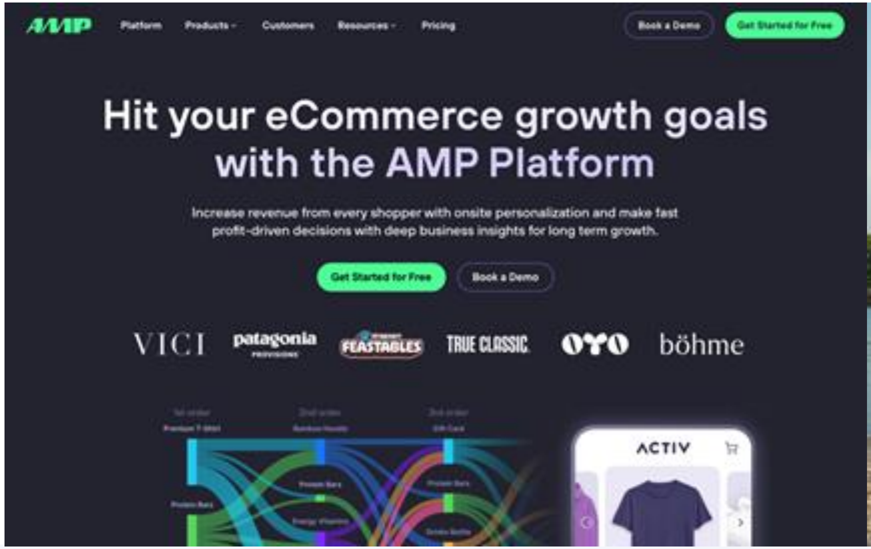
Love DnB, Psytrance, Scuba, MTB and Skiing!



AMP

AMP focuses on solving ecommerce problems

Team of ~30 based all over the world, mostly in APAC



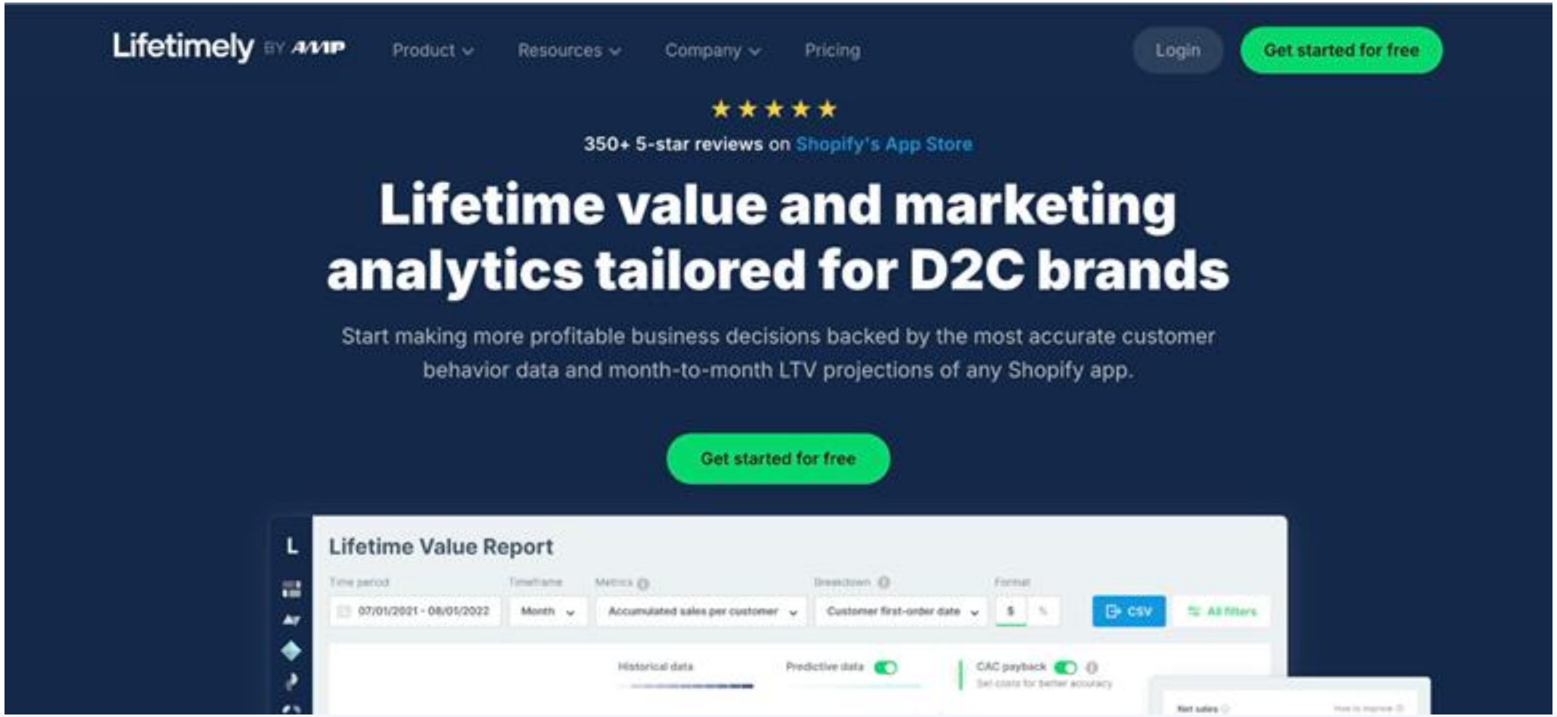
Hit your eCommerce growth goals with the AMP Platform

Increase revenue from every shopper with onsite personalization and make fast profit-driven decisions with deep business insights for long term growth.

Why ClickHouse?

Lifetimely

Our flagship analytics product



★★★★★

350+ 5-star reviews on Shopify's App Store

Lifetime value and marketing analytics tailored for D2C brands

Start making more profitable business decisions backed by the most accurate customer behavior data and month-to-month LTV projections of any Shopify app.

Lifetime Value Report

Time period: 07/01/2021 - 08/01/2022
Timeframe: Month
Metrics: Accumulated sales per customer
Breakdown: Customer first-order date
Format: $ %

How we got here

PostgreSQL

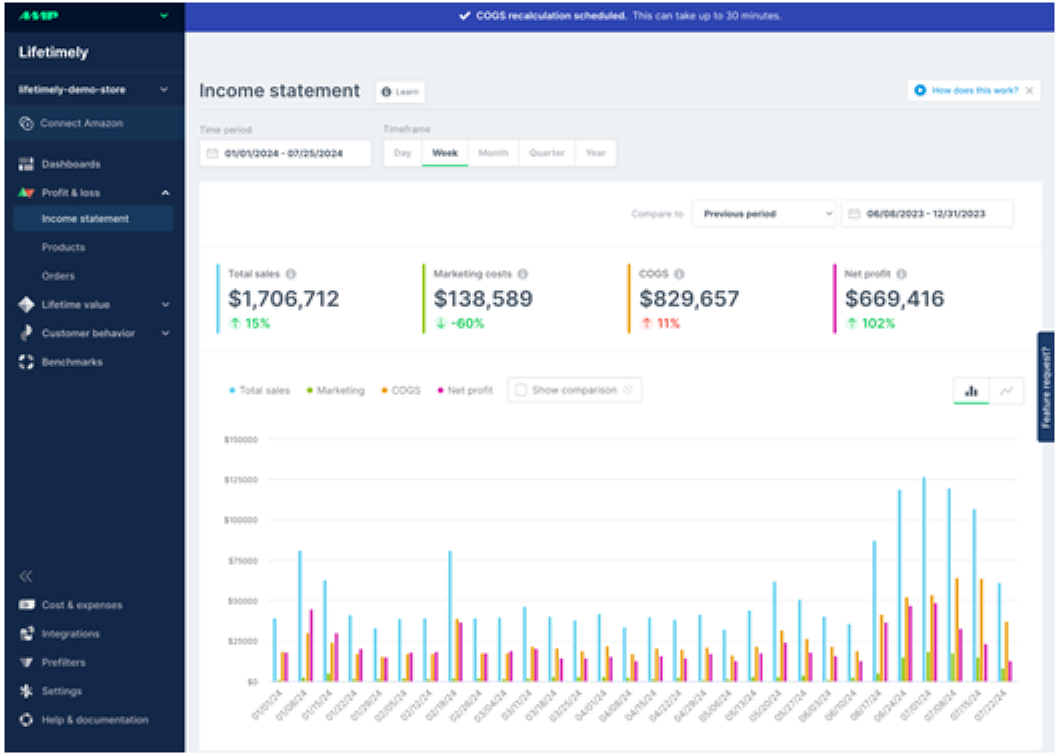
PG query performance degraded once over ~100m rows per table

Filters

Dynamic filtering requires query time computation for all reports

3,750+

Shopify stores ingesting and querying their data


COGS recalculation scheduled. This can take up to 30 minutes.

Income statement

Time period: 01/01/2024 - 07/25/2024

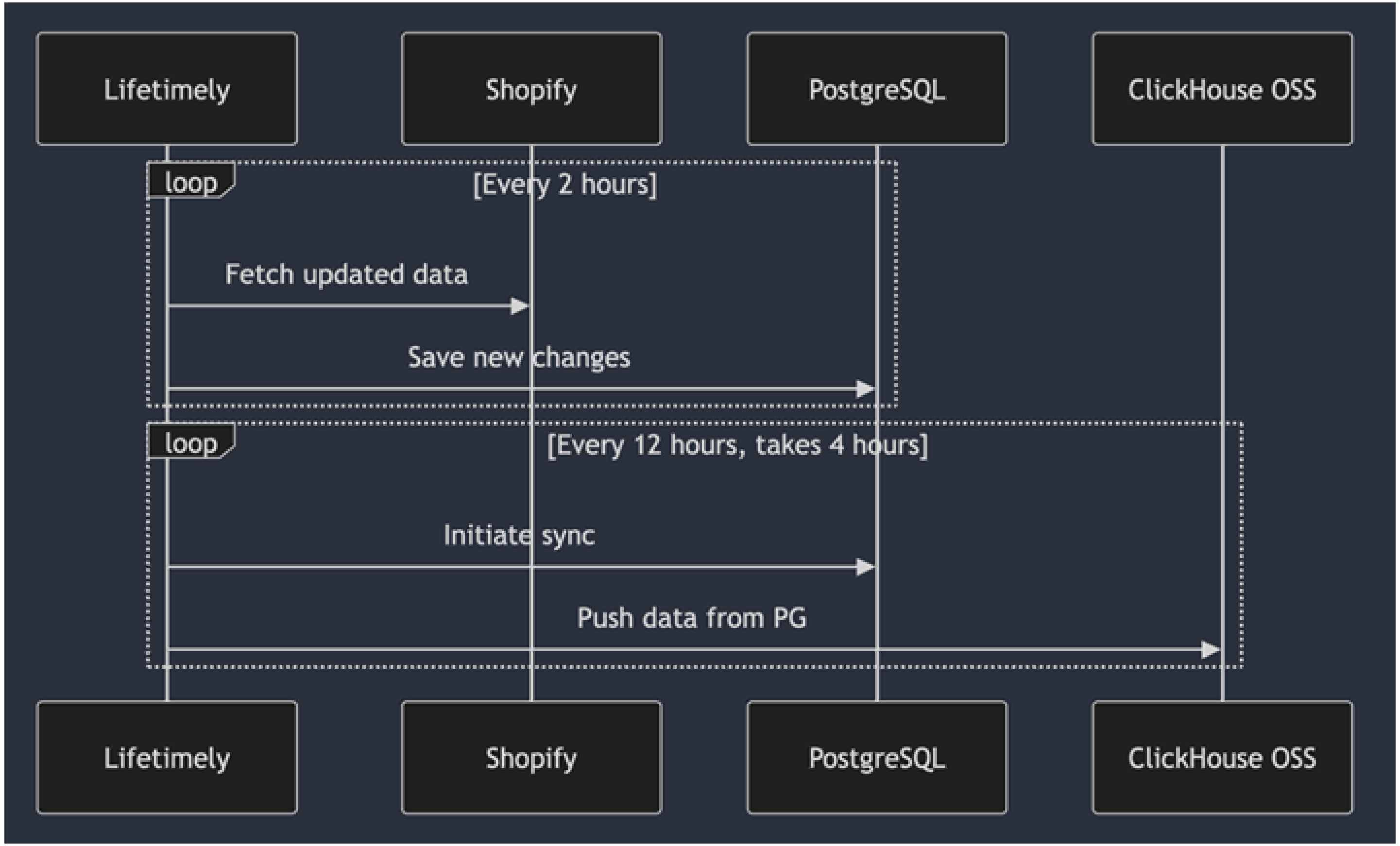
Timeframe: Day | Week | Month | Quarter | Year

Compare to: Previous period | 06/08/2023 - 12/31/2023

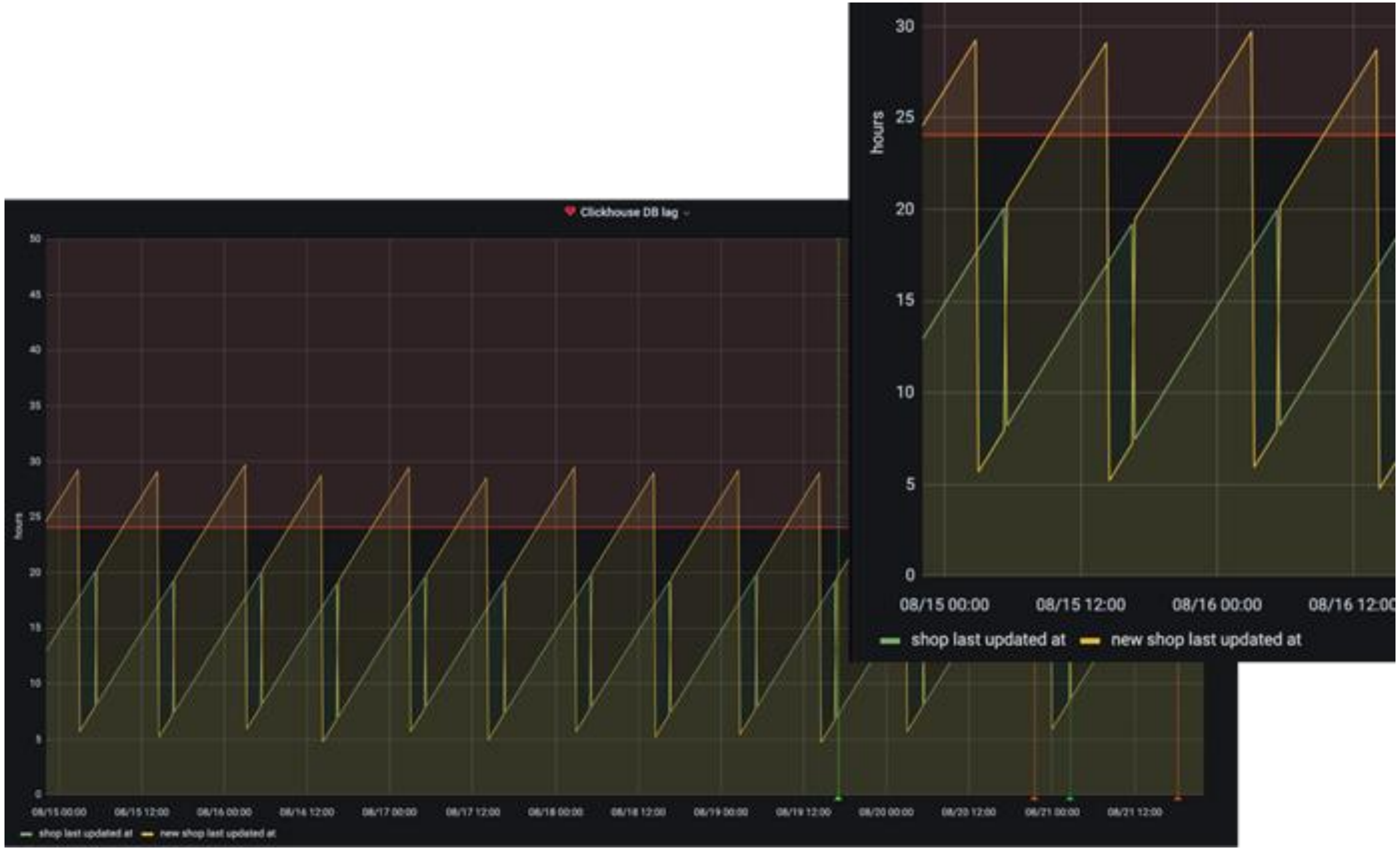
Total sales	Marketing costs	COGS	Net profit
$1,706,712	$138,589	$829,657	$669,416
↑ 15%	↓ -60%	↑ 11%	↑ 102%

“Lets clone our PG database to ClickHouse for queries”

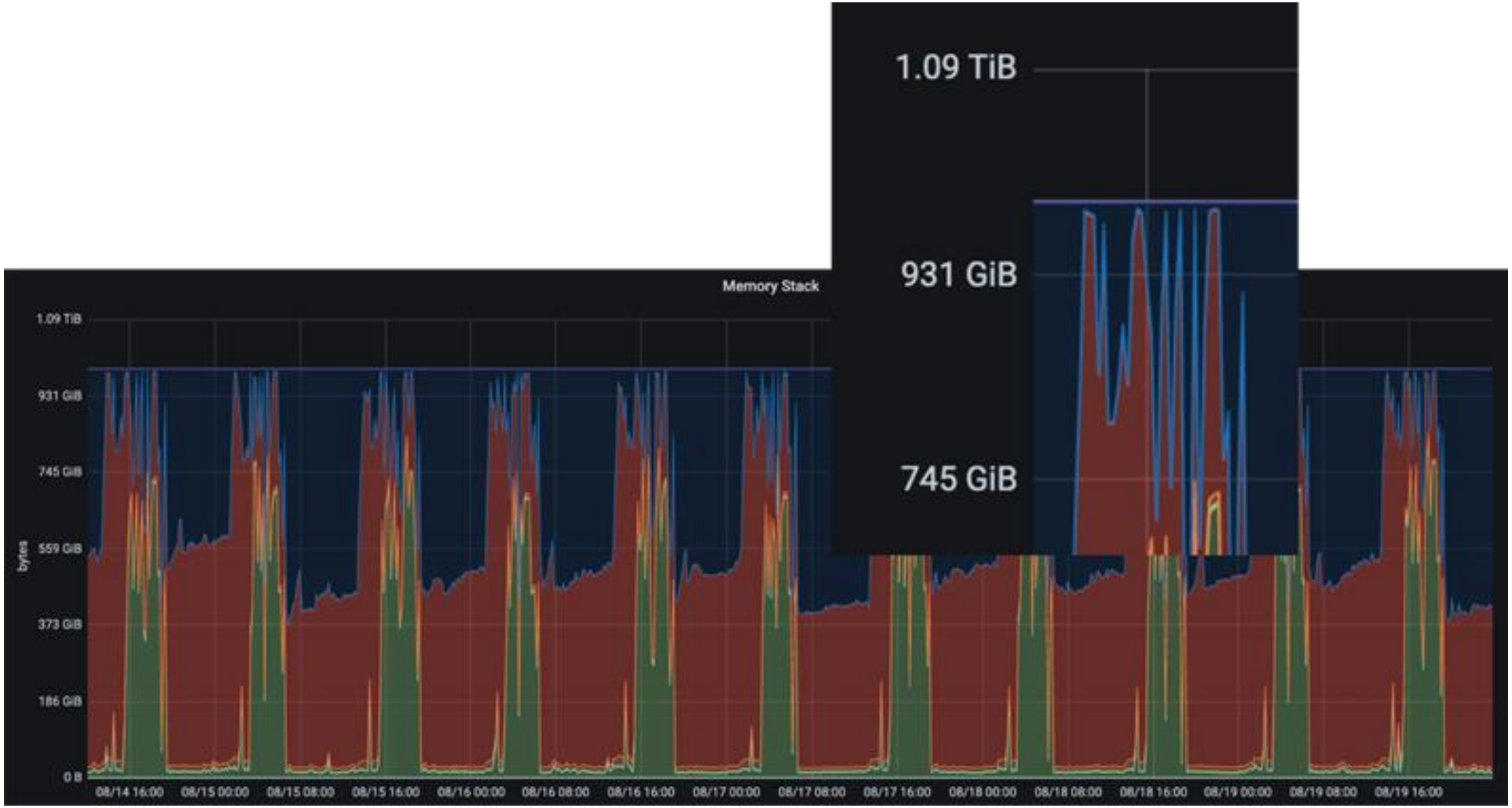
What problems are we facing?



Batch processing workflow



Problem 1 - Data Freshness



Problem 2 - Resource usage

Where do we want to go?

Our goals - Customer

Integrity

An observable and testable data pipeline

Freshness

Provide customer data in a realistic and timely manner

Flexibility

Enrich data with external integrations with ease

Our goals – Technical

Webhooks

Handle streamed events sent at us from Shopify

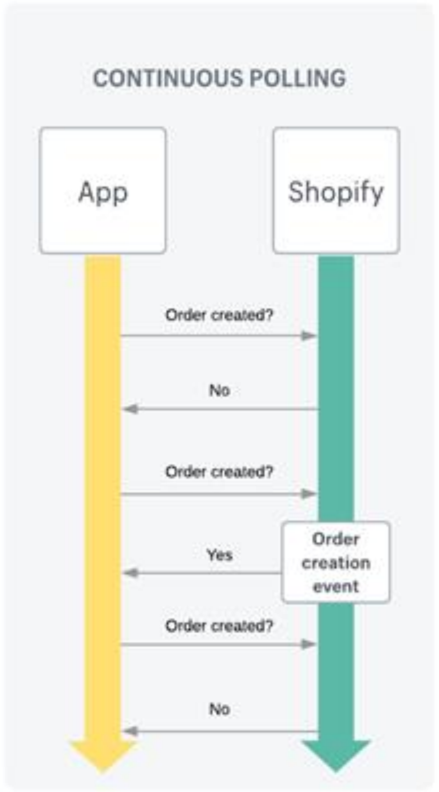
Replay

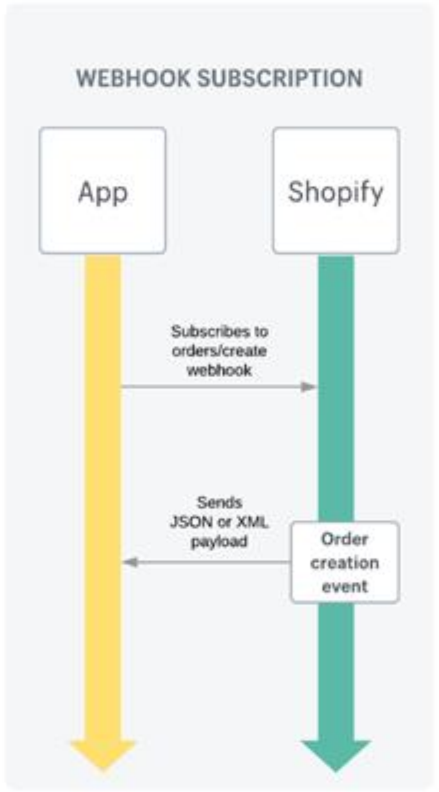
Store events in a data lake for backup and rebuild

IaC

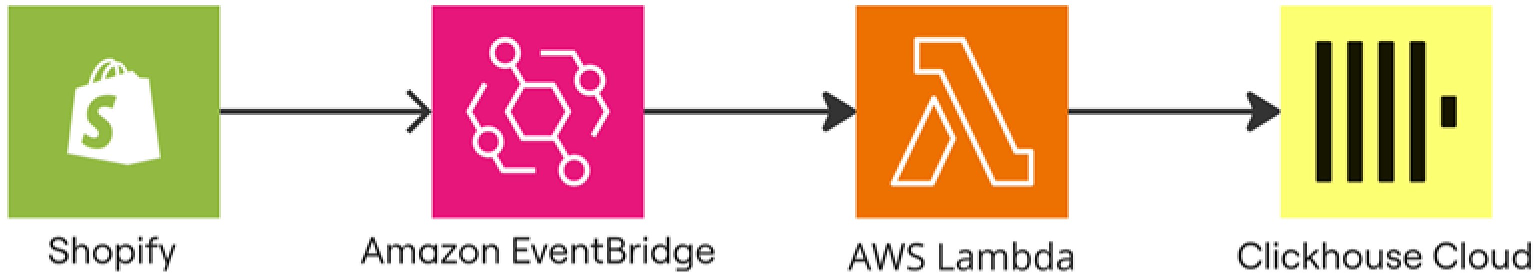
Define and release our infrastructure with Terraform

Let’s get streaming

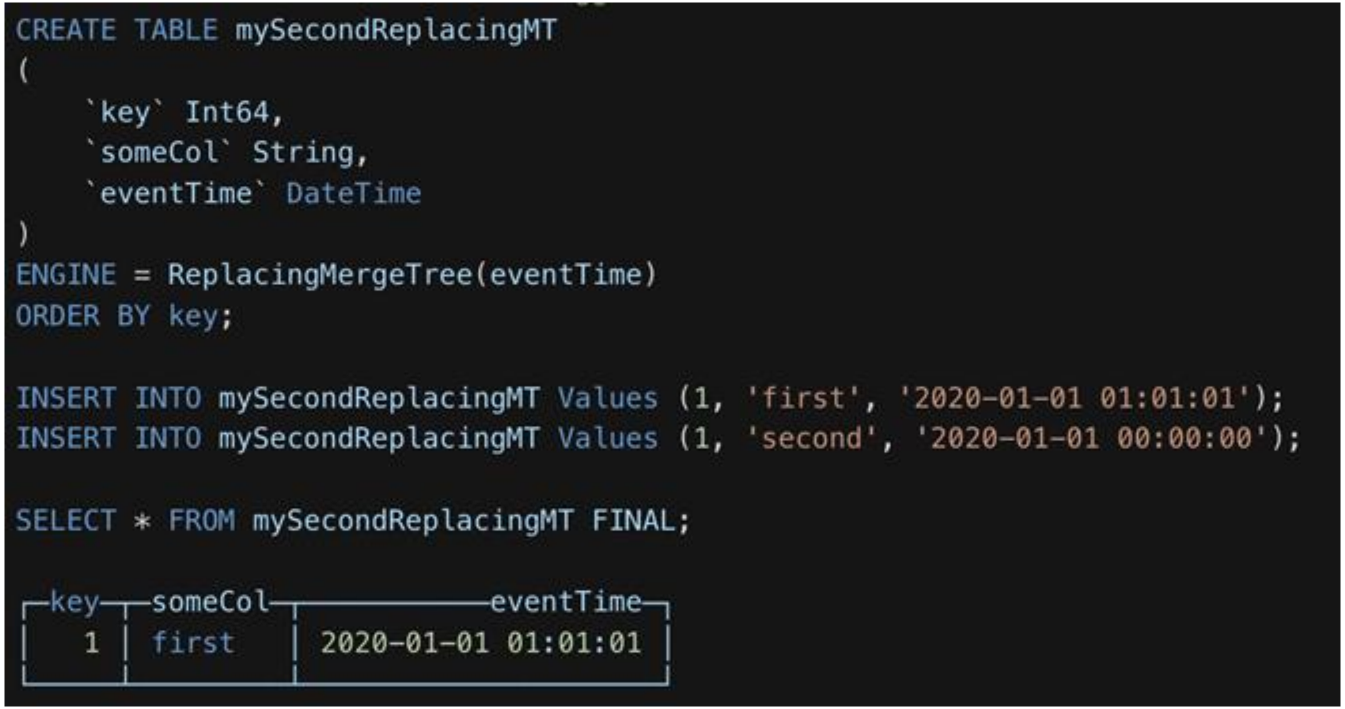



WEBHOOK SUBSCRIPTION
App
Shopify
Subscribes to orders/create webhook
Sends JSON or XML payload
Order creation event

Our data source - Shopify Webhooks


Shopify
Amazon EventBridge
AWS Lambda
Clickhouse Cloud

Solution 1: ReplacingMergeTree table engine



```
CREATE TABLE mySecondReplacingMT
(
    `key` Int64,
    `someCol` String,
    `eventTime` DateTime
)
ENGINE = ReplacingMergeTree(eventTime)
ORDER BY key;

INSERT INTO mySecondReplacingMT Values (1, 'first', '2020-01-01 01:01:01');
INSERT INTO mySecondReplacingMT Values (1, 'second', '2020-01-01 00:00:00');

SELECT * FROM mySecondReplacingMT FINAL;

┌─key─┬─someCol─┬───────────eventTime─┐
│   1 │ first   │ 2020-01-01 01:01:01 │
└─────┴─────────┴─────────────────────┘
```

“We can insert every update and let ClickHouse handle deduplication”

“We can insert every update and let ClickHouse handle deduplication”

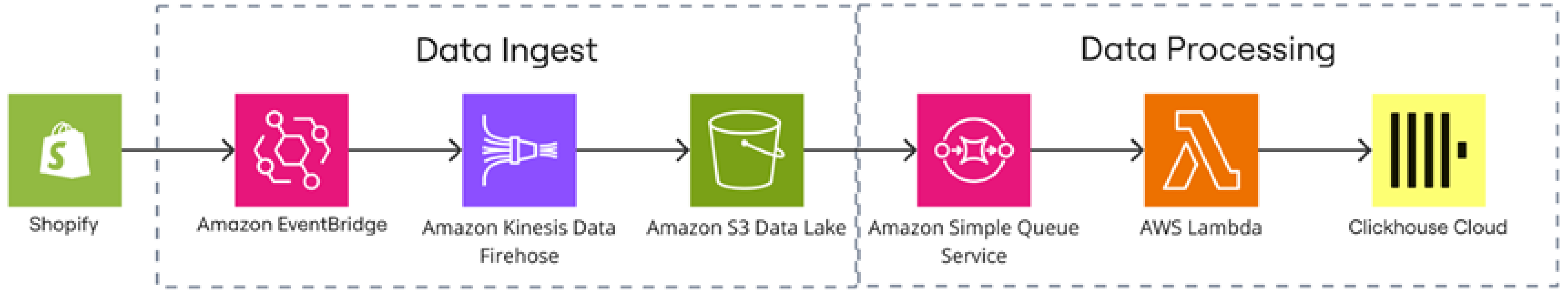
Buuuuut, we have a problem…

“Generally, we recommend inserting data in fairly large batches of at least 1,000 rows at a time, and ideally between 10,000 to 100,000 rows.”

- ClickHouse Docs

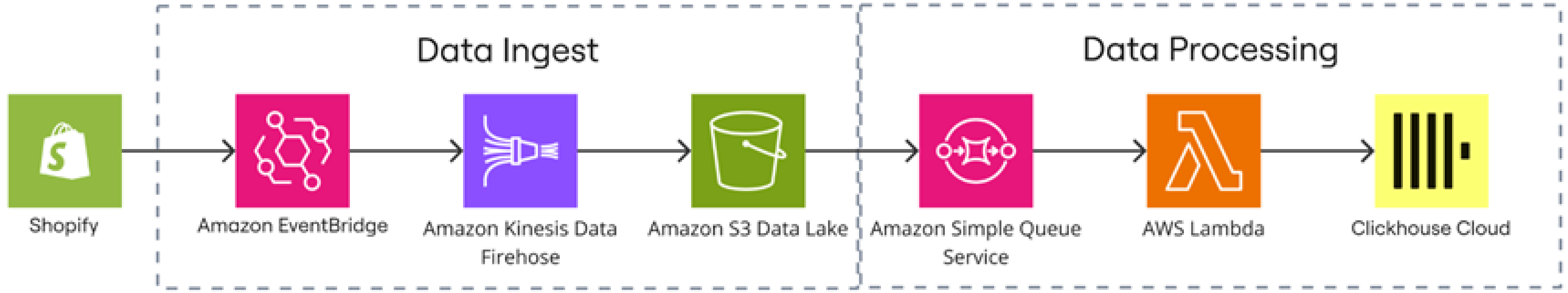
https://clickhouse.com/docs/en/cloud/bestpractices/bulk-inserts

Solution 2: AWS Data Firehose



Data Firehose allows buffering of JSON files into S3

Solution 2: AWS Data Firehose

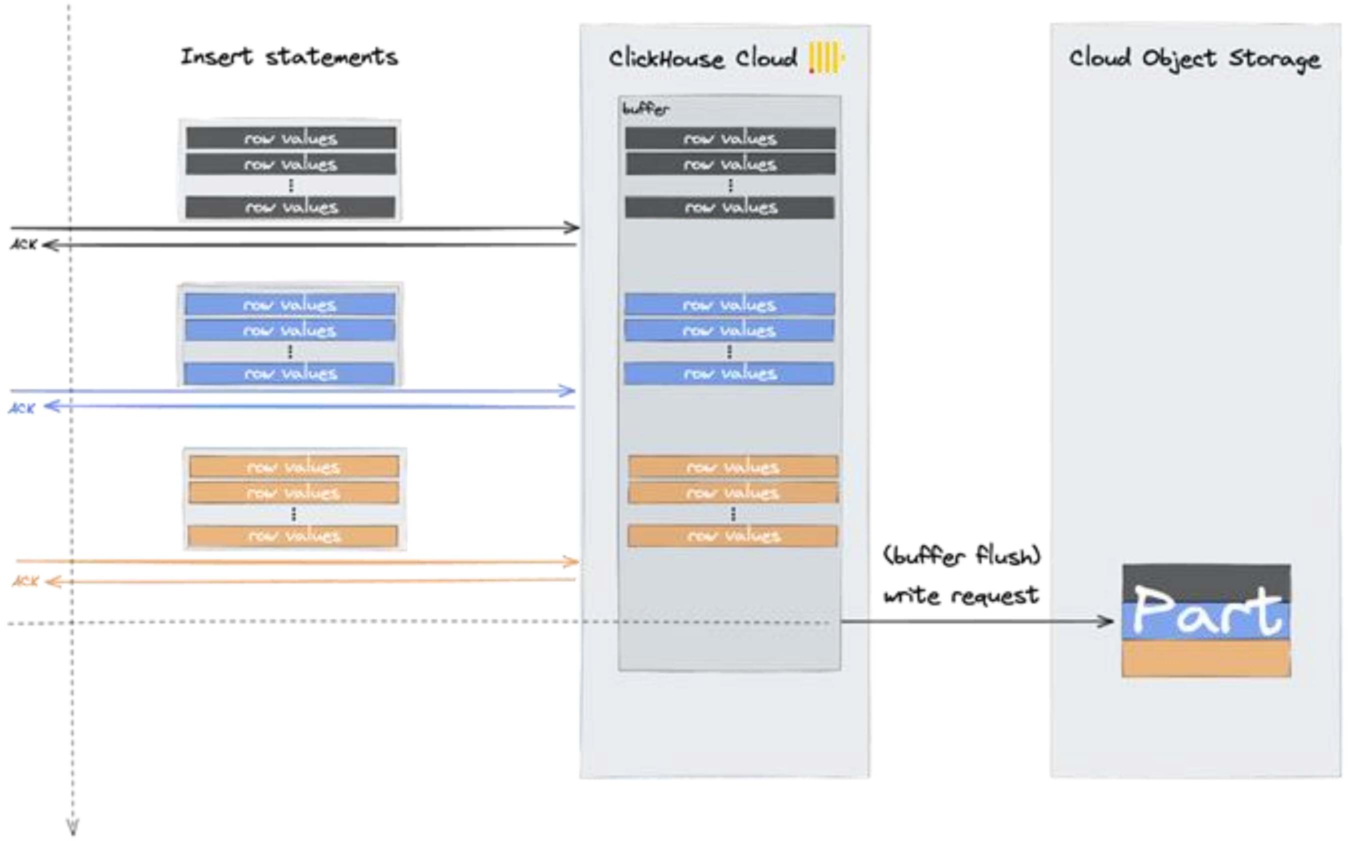


Data Firehose allows buffering of JSON files into S3

Buuuuut, we have another problem...

“Our buffers are still under 1,000 rows”

Solution 3: Async Inserts



Solution 3: Async Inserts

Inserts

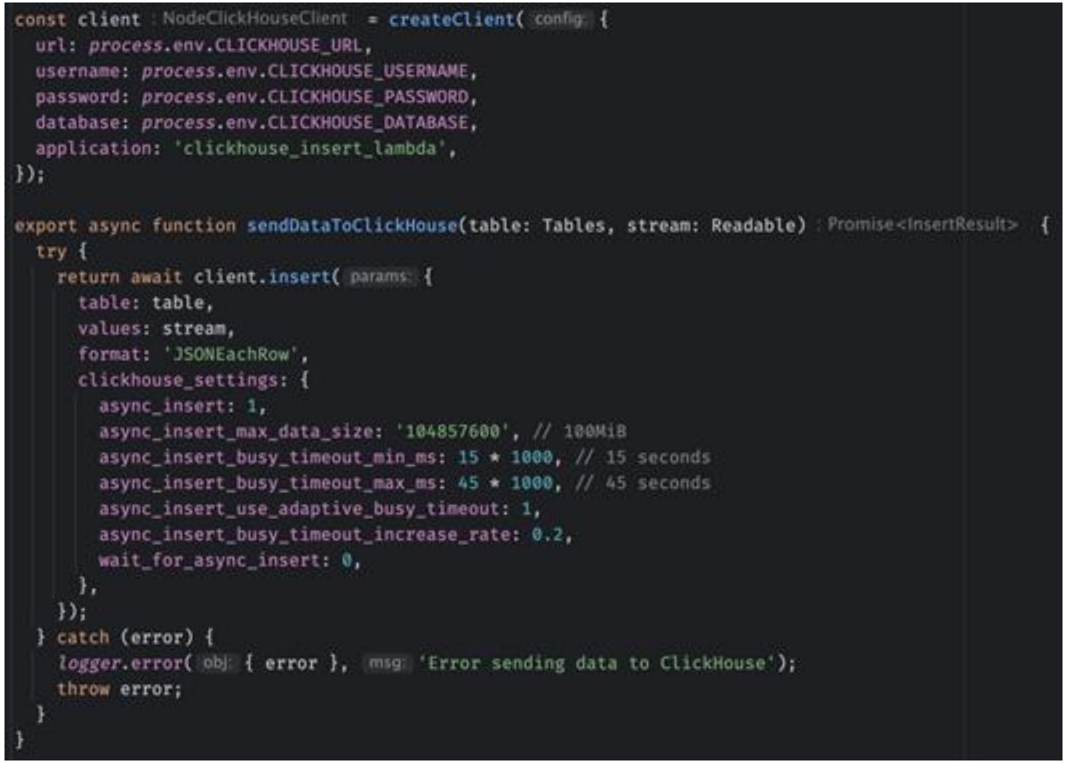
Send insert of any size into ClickHouse

Memory

Inserts are stored in memory

Flush

Inserts flushed to disk when parameters are reached

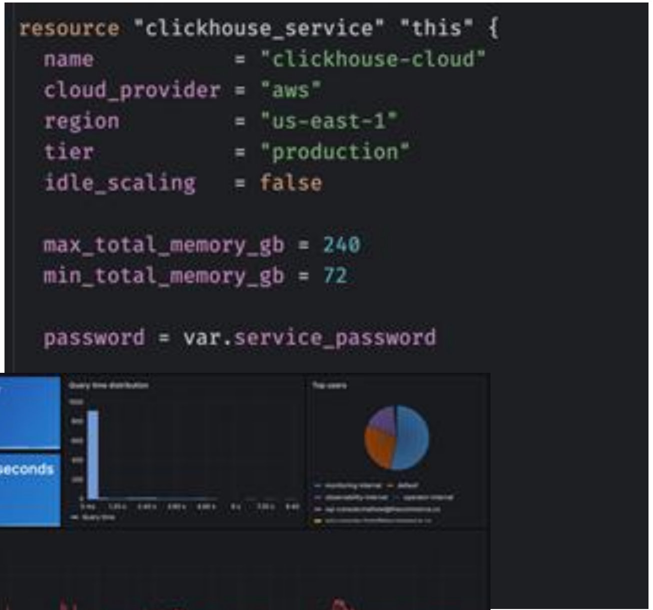


```
const client : NodeClickHouseClient  = createClient( config: {
  url: process.env.CLICKHOUSE_URL,
  username: process.env.CLICKHOUSE_USERNAME,
  password: process.env.CLICKHOUSE_PASSWORD,
  database: process.env.CLICKHOUSE_DATABASE,
  application: 'clickhouse_insert_lambda',
});

export async function sendDataToClickHouse(table: Tables, stream: Readable) : Promise<InsertResult>  {
  try {
    return await client.insert( params: {
      table: table,
      values: stream,
      format: 'JSONEachRow',
      clickhouse_settings: {
        async_insert: 1,
        async_insert_max_data_size: '104857600', // 100MiB
        async_insert_busy_timeout_min_ms: 15 * 1000, // 15 seconds
        async_insert_busy_timeout_max_ms: 45 * 1000, // 45 seconds
        async_insert_use_adaptive_busy_timeout: 1,
        async_insert_busy_timeout_increase_rate: 0.2,
        wait_for_async_insert: 0,
      },
    });
  } catch (error) {
    logger.error( obj: { error }, msg: 'Error sending data to ClickHouse');
    throw error;
  }
}
```

Solution 4: ClickHouse Cloud Terraform provider

- Create reproducible ClickHouse instances across all our environments
- Spin up services in CICD for testing
- Monitoring and automatic updates
- Autoscaling



```
resource "clickhouse_service" "this" {
  name           = "clickhouse-cloud"
  cloud_provider = "aws"
  region         = "us-east-1"
  tier           = "production"
  idle_scaling   = false

  max_total_memory_gb = 240
  min_total_memory_gb = 72

  password = var.service_password
```

Demo!