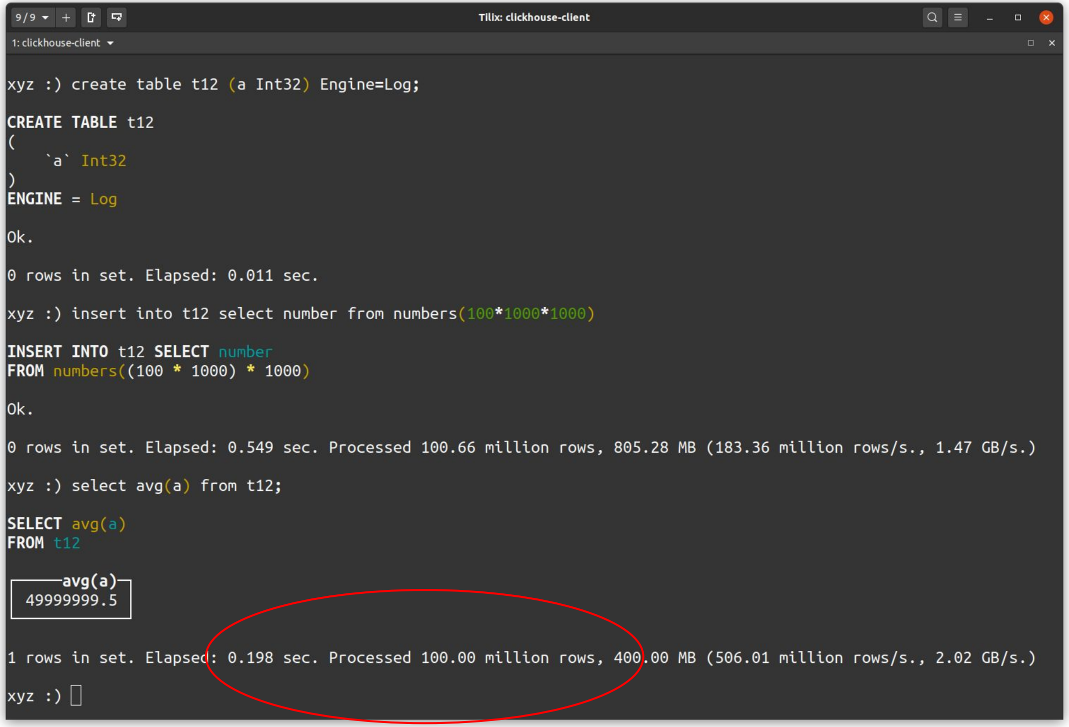
# How we processed 12 Trillion Rows during Black Friday

tinybird.co - @javisantana

12,213,675,632,435



```
xyz :) create table t12 (a Int32) Engine=Log;

CREATE TABLE t12
(
    `a` Int32
)
ENGINE = Log

Ok.

0 rows in set. Elapsed: 0.011 sec.

xyz :) insert into t12 select number from numbers(100*1000*1000)

INSERT INTO t12 SELECT number
FROM numbers((100 * 1000) * 1000)

Ok.

0 rows in set. Elapsed: 0.549 sec. Processed 100.66 million rows, 805.28 MB (183.36 million rows/s., 1.47 GB/s.)

xyz :) select avg(a) from t12;

SELECT avg(a)
FROM t12

┌─avg(a)─┐
│ 49999999.5 │
└────────┘

1 rows in set. Elapsed: 0.198 sec. Processed 100.00 million rows, 400.00 MB (506.01 million rows/s., 2.02 GB/s.)

xyz :)
```

## 12T - ~6 hours

(modest laptop)

## The problem

Tinybird

SaaS product to build real time data products with high amounts of data

Not the sales guy here, but this gives context to this talk

4 weeks before Black Friday

Client [who does not want to who they are] in the retail space:

“Hey, we have a project for you…”

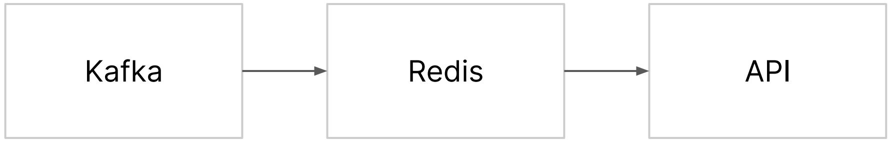
## The use case

Report sales in real time for different countries for different areas in the company.

Our job: expose an API to feed their dashboards

The use case

That’s easy



The use case: but wait, here is the reality

This is not a pretty dashboard with big numbers to post screenshots on twitter

## The use case: but wait, here is the reality

Data origin: [legacy] transactional database (aka no change events, no CDC)

500+ concurrent users during that night

Real time

Multiple filters and configurable options, not just global counters

## The use case: the data stream

Events with sales: product, units, amount... 12 columns

5 different data sources

5 batches per second with the last 5 minutes of data: lot of duplicated rows

## The use case: the problems

Lot of read concurrency

Millions of rows per minute input

Deduplicate those rows

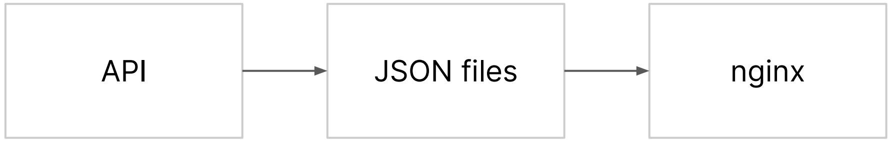
## Ok, what’s the plan?

## The approach: ideally

Serve static JSON files generated from the input.

Cheap, scalable and simple

Way too many different combinations (filters + configuration)

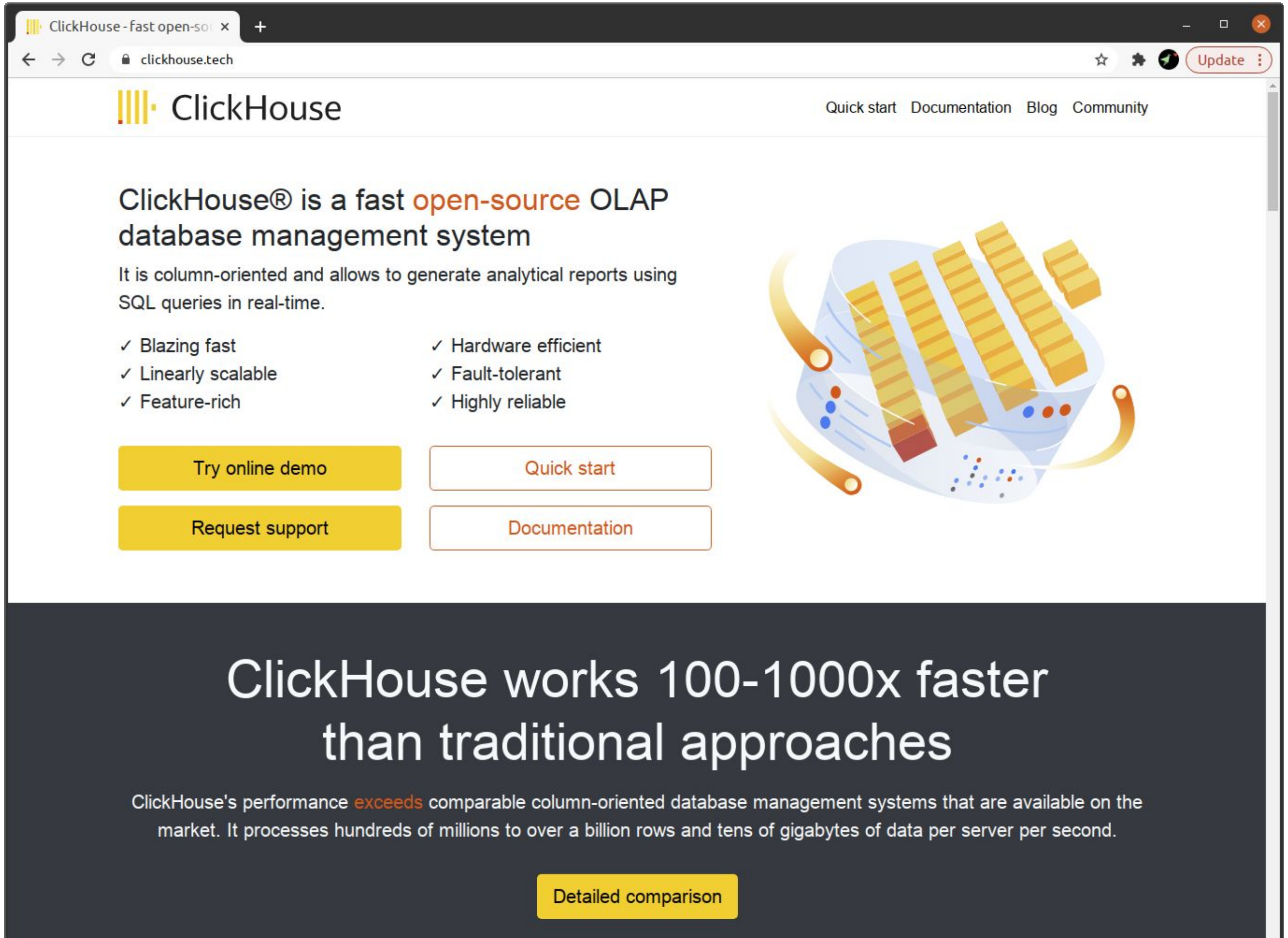


## The approach: the decision, some background

We are boring, we tend to use technology with 15+ years. Also we did this in the past in previous companies

No actual time to make complex decisions and our product solves this problem

We use Clickhouse in our product since 2018 and looks a good fit



ClickHouse® is a fast open-source OLAP database management system

It is column-oriented and allows to generate analytical reports using SQL queries in real-time.

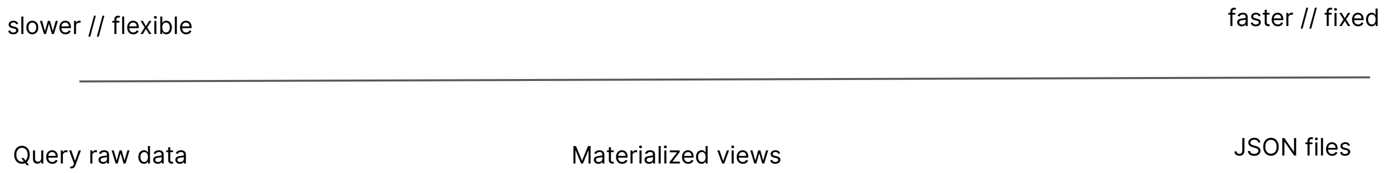
| | |
|---|---|
| ✓ Blazing fast | ✓ Hardware efficient |
| ✓ Linearly scalable | ✓ Fault-tolerant |
| ✓ Feature-rich | ✓ Highly reliable |

ClickHouse works 100-1000x faster than traditional approaches

ClickHouse's performance exceeds comparable column-oriented database management systems that are available on the market. It processes hundreds of millions to over a billion rows and tens of gigabytes of data per server per second.

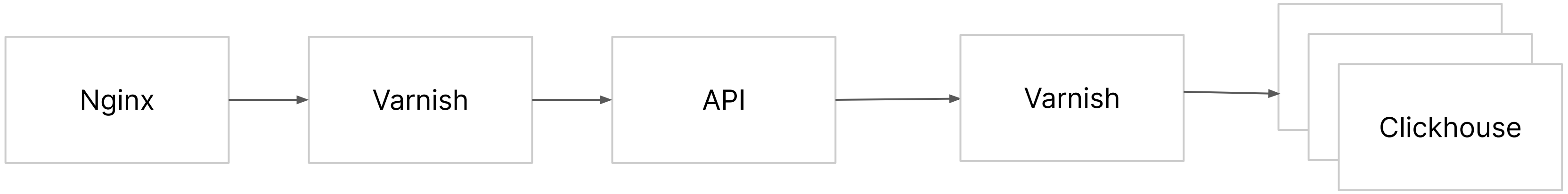
## The approach: the decision

Clickhouse has a nice feature: [Incremental] Materialized views



## The solution

## Infrastructure to handle 200-300QPS



Just https

Load balancing

Grace mode

Python

Data import
API Endpoints

Load balancing

Health checks

Database engine

(+ zookeeper)

The solution: first problem, dealing with import batches

Kafka ?

Data is sent in batches

Append only “landing” table leveraging Clickhouse super fast data import (also multimaster)

Generate all the views based on that “landing” table

The solution: dealing with import data

Upserts - clickhouse is not the best one here

30 minutes window to upsert

Solution: real time + historic tables

2 MV (for RT and historic) to cover 90% the API endpoints

Simple, fast to generate MV

The solution: some small but import details

Deal with replication lag

Materialized views must fit in memory

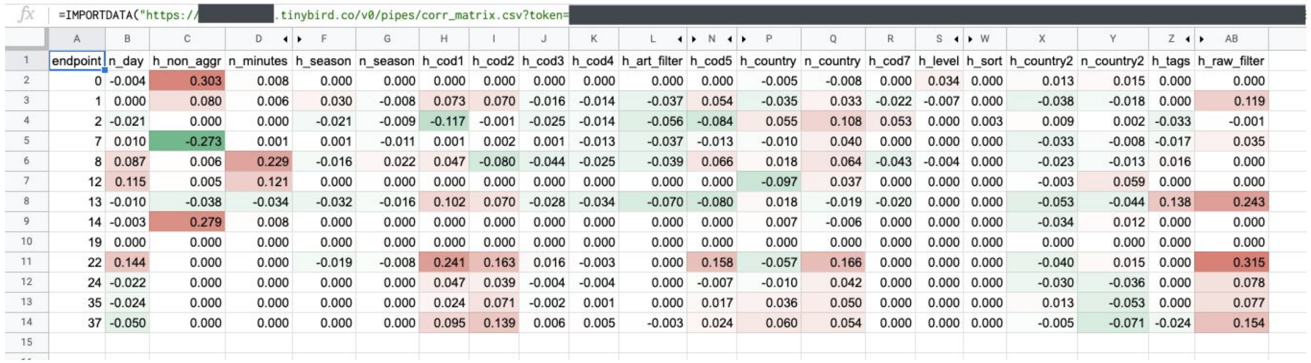
Leverage caching and indexing as much as possible

The solution: endpoints, aka queries to Clickhouse

Objective: 1 core - 150ms (q95)

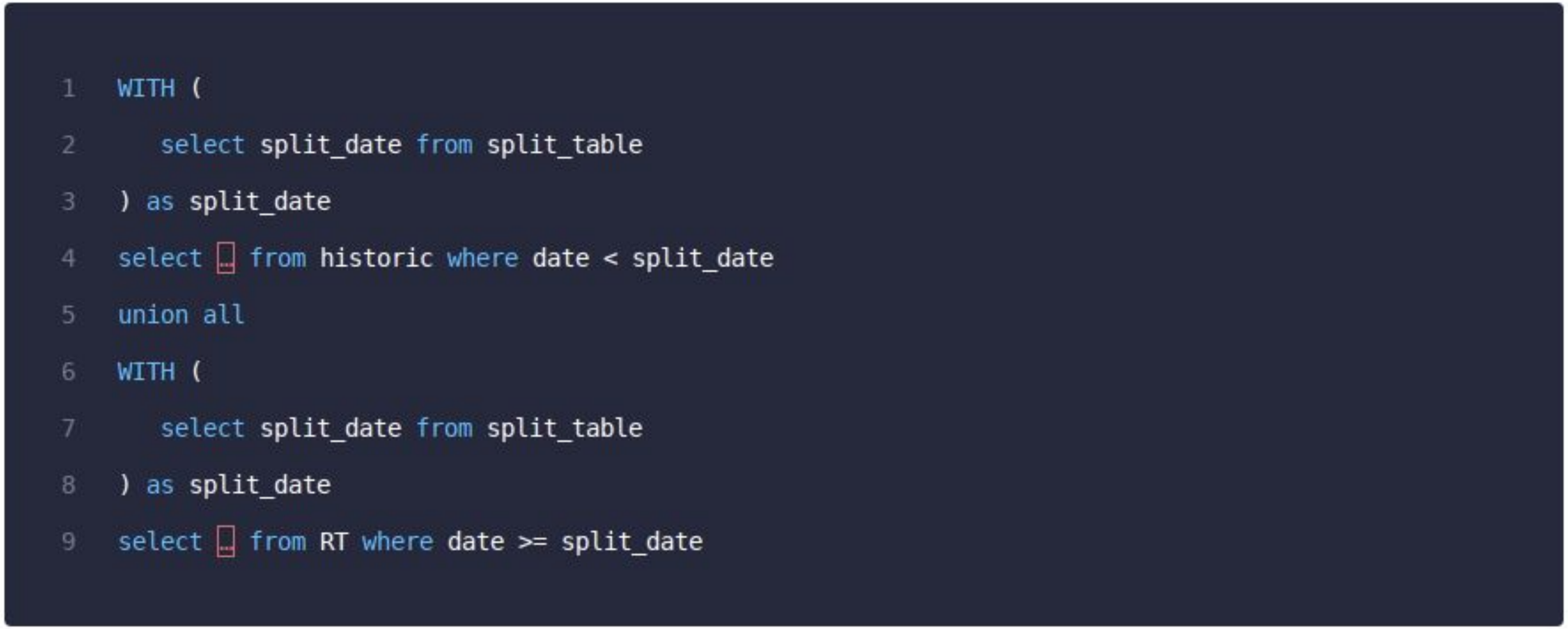
Materialize data taking into account the (estimated) call distribution

Data driven optimization: a simple spreadsheet



```
=IMPORTDATA("https://[redacted].tinybird.co/v0/pipes/corr_matrix.csv?token=[redacted]
```

| | A | B | C | D | F | G | H | I | J | K | L | N | P | Q | R | S | W | X | Y | Z | AB |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | endpoint | n_day | h_non_aggr | n_minutes | h_season | n_season | h_cod1 | h_cod2 | h_cod3 | h_cod4 | h_art_filter | h_cod5 | h_country | n_country | h_cod7 | h_level | h_sort | h_country2 | n_country2 | h_tags | h_raw_filter |
| 2 | 0 | -0.004 | 0.303 | 0.008 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | -0.005 | -0.008 | 0.000 | 0.034 | 0.000 | 0.013 | 0.015 | 0.000 | 0.000 |
| 3 | 1 | 0.000 | 0.080 | 0.006 | 0.030 | -0.008 | 0.073 | 0.070 | -0.016 | -0.014 | -0.037 | 0.054 | -0.035 | 0.033 | -0.022 | -0.007 | 0.000 | -0.038 | -0.018 | 0.000 | 0.119 |
| 4 | 2 | -0.021 | 0.000 | 0.000 | -0.021 | -0.009 | -0.117 | -0.001 | -0.025 | -0.014 | -0.056 | -0.084 | 0.055 | 0.108 | 0.053 | 0.000 | 0.003 | 0.009 | 0.002 | -0.033 | -0.001 |
| 5 | 7 | 0.010 | -0.273 | 0.001 | 0.001 | -0.011 | 0.001 | 0.002 | 0.001 | -0.013 | -0.037 | -0.013 | -0.010 | 0.040 | 0.000 | 0.000 | 0.000 | -0.033 | -0.008 | -0.017 | 0.035 |
| 6 | 8 | 0.087 | 0.006 | 0.229 | -0.016 | 0.022 | 0.047 | -0.080 | -0.044 | -0.025 | -0.039 | 0.066 | 0.018 | 0.064 | -0.043 | -0.004 | 0.000 | -0.023 | -0.013 | 0.016 | 0.000 |
| 7 | 12 | 0.115 | 0.005 | 0.121 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | -0.097 | 0.037 | 0.000 | 0.000 | 0.000 | -0.003 | 0.059 | 0.000 | 0.000 |
| 8 | 13 | -0.010 | -0.038 | -0.034 | -0.032 | -0.016 | 0.102 | 0.070 | -0.028 | -0.034 | -0.070 | -0.080 | 0.018 | -0.019 | -0.020 | 0.000 | 0.000 | -0.053 | -0.044 | 0.138 | 0.243 |
| 9 | 14 | -0.003 | 0.279 | 0.008 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.007 | -0.006 | 0.000 | 0.000 | 0.000 | -0.034 | 0.012 | 0.000 | 0.000 |
| 10 | 19 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 11 | 22 | 0.144 | 0.000 | 0.000 | -0.019 | -0.008 | 0.241 | 0.163 | 0.016 | -0.003 | 0.000 | 0.158 | -0.057 | 0.166 | 0.000 | 0.000 | 0.000 | -0.040 | 0.015 | 0.000 | 0.315 |
| 12 | 24 | -0.022 | 0.000 | 0.000 | 0.000 | 0.000 | 0.047 | 0.039 | -0.004 | -0.004 | 0.000 | -0.007 | -0.010 | 0.042 | 0.000 | 0.000 | 0.000 | -0.030 | -0.036 | 0.000 | 0.078 |
| 13 | 35 | -0.024 | 0.000 | 0.000 | 0.000 | 0.000 | 0.024 | 0.071 | -0.002 | 0.001 | 0.000 | 0.017 | 0.036 | 0.050 | 0.000 | 0.000 | 0.000 | 0.013 | -0.053 | 0.000 | 0.077 |
| 14 | 37 | -0.050 | 0.000 | 0.000 | 0.000 | 0.000 | 0.095 | 0.139 | 0.006 | 0.005 | -0.003 | 0.024 | 0.060 | 0.054 | 0.000 | 0.000 | 0.000 | -0.005 | -0.071 | -0.024 | 0.154 |
| 15 | | | | | | | | | | | | | | | | | | | | | |



```
WITH (
   select split_date from split_table
) as split_date
select … from historic where date < split_date
union all
WITH (
   select split_date from split_table
) as split_date
select … from RT where date >= split_date
```

Black Friday day[s]

650B rows ingested

12T rows queried

50QPS median, 300QPS peak

600ms q95 response time by the end of the BF :(

## Thank you!

https://blog.tinybird.co/2020/12/21/how-we-setup-real-time-analytics-service-to-process-12-trillion-rows-during-black-friday/

https://tinybird.co

@javisantana