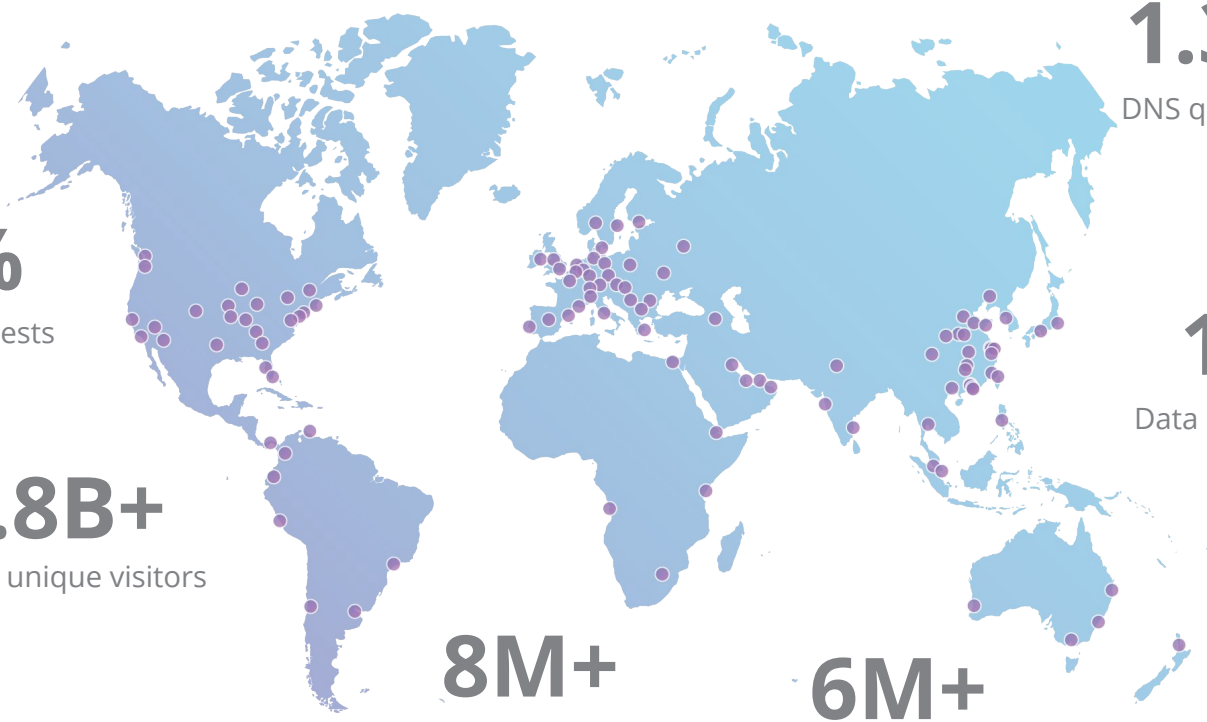
HTTP Analytics for 6M requests per second using ClickHouse

Alexander Bocharov

Cloudflare scale



1.3M+
DNS queries/second

10%
Internet requests everyday

150+
Data centers globally

2.8B+
Monthly unique visitors

8M+
websites, apps & APIs in 150 countries

6M+
HTTP requests/second

CLOUDFLARE

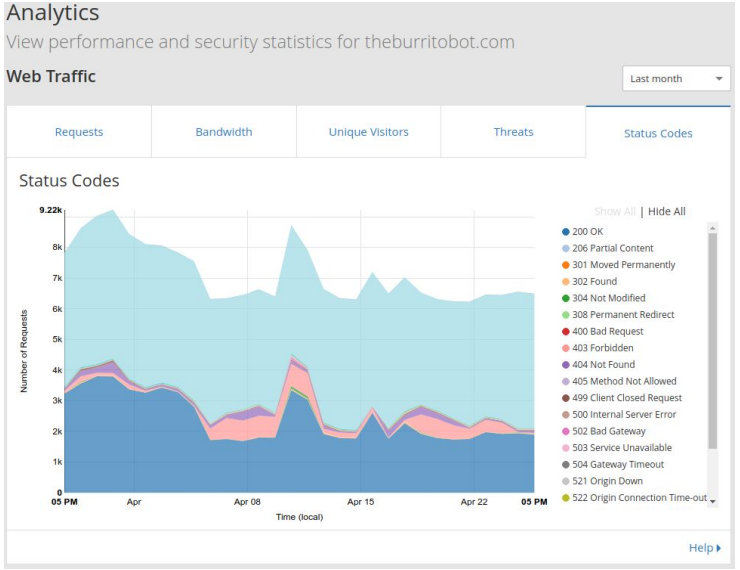
Cloudflare use cases

- ➔ HTTP analytics
- ➔ DNS analytics
- ➔ Argo analytics
- ➔ Netflow analytics
- ➔ DNS Resolver analytics
- ➔ Recursor analytics
- ➔ Beacons analytics
- ➔ Edge workers analytics
- ➔ DDoS attacks analysis
- ➔ Internal BI
- ➔ More at the end...

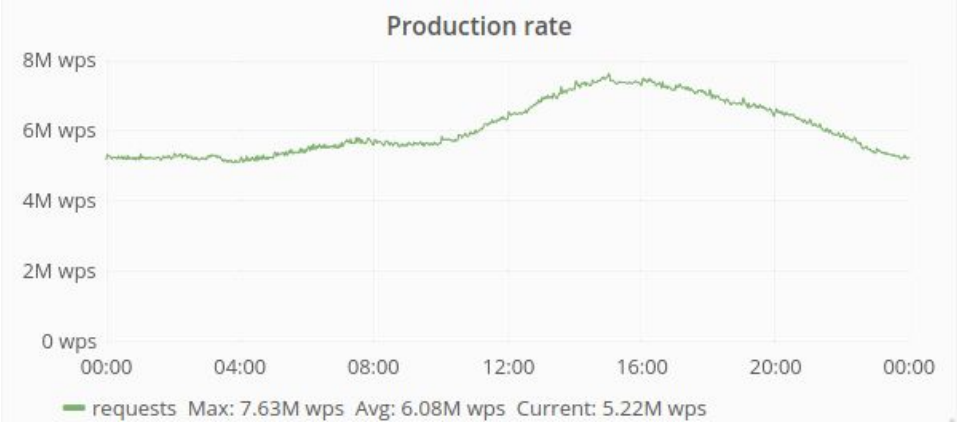
CLOUDFLARE

HTTP Analytics use case

- ➔ Analytics tab in Cloudflare UI dashboard
- ➔ Zone Analytics API
  - ◆ Dashboard endpoint
  - ◆ Co-locations endpoint (Enterprise plan only)
- ➔ Internal BI tools

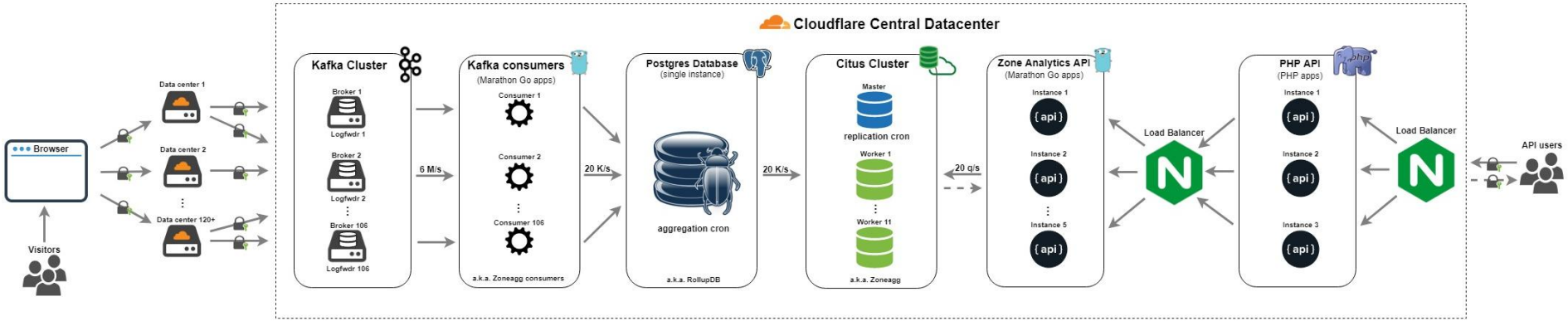


- ➔ 6M HTTP requests per second
- ➔ 1630B Kafka request message size
- ➔ 150+ fields in request message



CLOUDFLARE

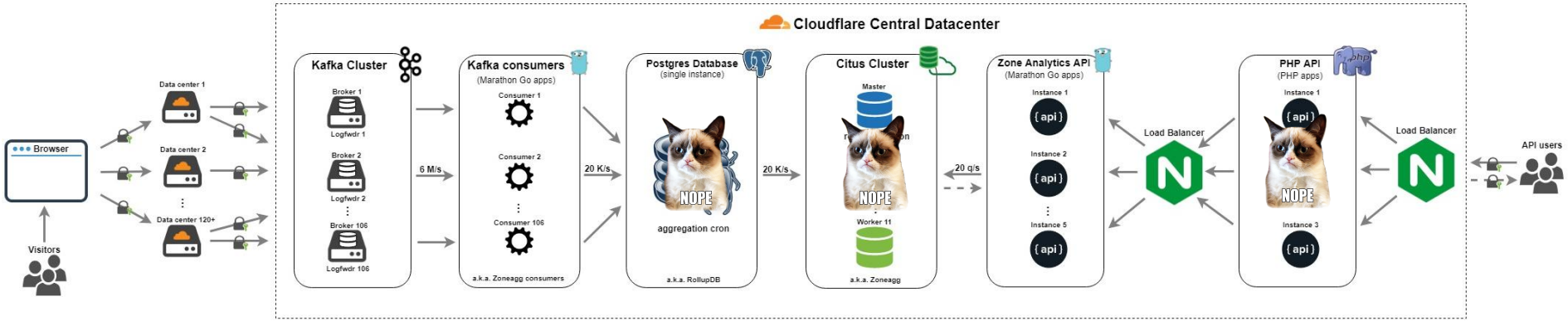
Previous pipeline (2014-2018)



- ➔ Log Forwarder
- ➔ Kafka cluster
- ➔ Kafka consumers
- ➔ Postgres database
- ➔ Citus Cluster
- ➔ Zone Analytics API
- ➔ PHP API
- ➔ Load Balancer

CLOUDFLARE

Previous pipeline issues

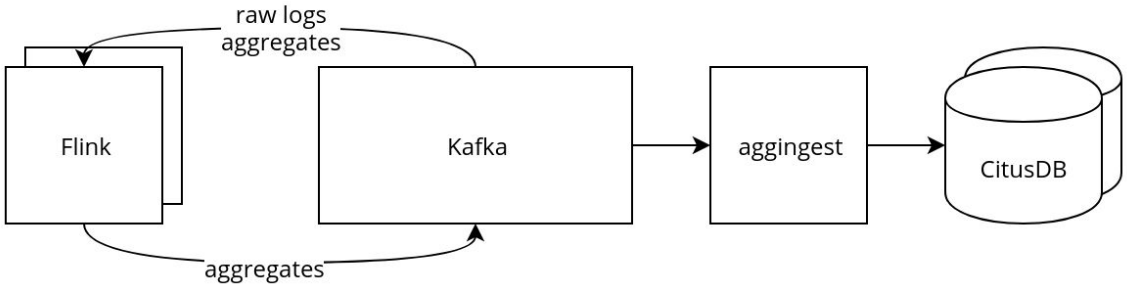


- ➔ Postgres SPoF
- ➔ Citus master SPoF
- ➔ Complex codebase

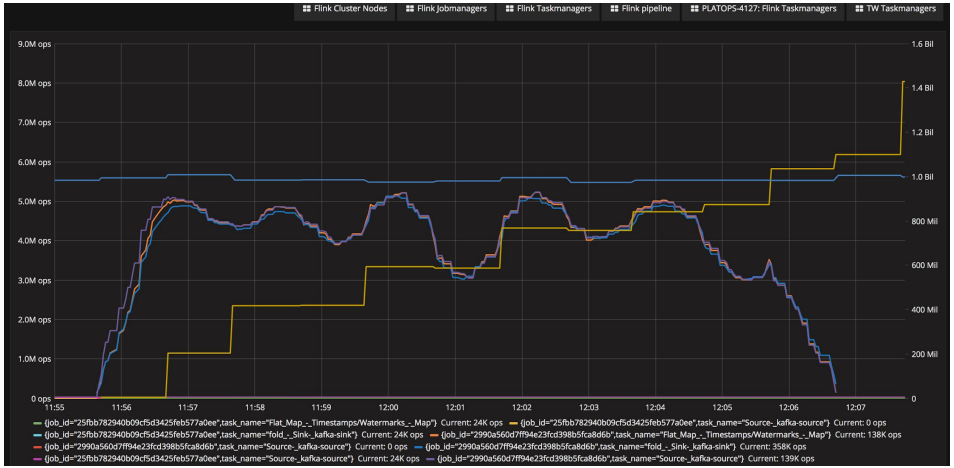
- ➔ Many dependencies
- ➔ High maintenance cost
- ➔ Doesn't scale

CLOUDFLARE

Flink attempt

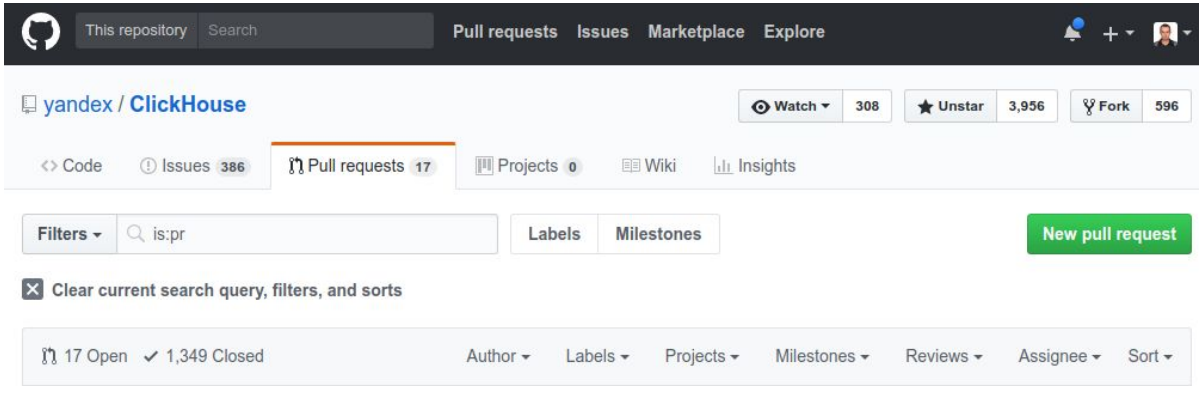


- ➔ Worked well with smaller volume
- ➔ Major key zoneId skew
- ➔ The fold operation is one of bottlenecks
- ➔ Checkpointing degrades performance
- ➔ Flink poor debuggability
- ➔ Time to market is too high



CLOUDFLARE

ClickHouse



- ➔ Open source
- ➔ Blazing fast
- ➔ Linearly scalable
- ➔ Fault tolerant
- ➔ Availability in CAP
- ➔ Responsive community

- ➔ Existing in-house expertise
- ➔ Excellent documentation
- ➔ Easy to work with
- ➔ SQL!!!

CLOUDFLARE

DNS vs HTTP pipeline

DNS (rrdns)

- ➔ 1.3M messages / sec
- ➔ 130B per message
- ➔ 40 fields

HTTP

- ➔ 6M messages / sec
- ➔ 1630B per message
- ➔ 130+ fields

blog.cloudflare.com/how-cloudflare-analyzes-1m-dns-queries-per-second

CLOUDFLARE

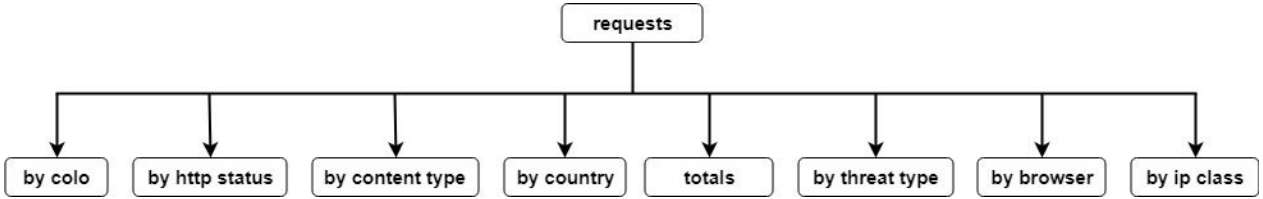
ClickHouse schema design

- ➔ Compatibility with Citus schema
- ➔ Possibility to migrate data from Citus
- ➔ Space efficient
- ➔ Raw data is not an option!

Metric	Cap'n Proto	Cap'n Proto (zstd)	ClickHouse
Avg message/record size	1630 B	360 B	36.74 B
Storage requirements for 1 year	273.93 PiB	60.5 PiB	18.52 PiB (RF x3)
Storage cost for 1 year	$28M	$6.2M	$1.9M

CLOUDFLARE

Aggregations schema design #1



Materialized view for each of the breakdowns:

- ➔ Totals (requests, bytes, threats, unique IPs, etc.)
- ➔ Breakdown by colo ID
- ➔ Breakdown by HTTP status
- ➔ Breakdown by content type
- ➔ Breakdown by country
- ➔ Breakdown by threat type
- ➔ Breakdown by browser type
- ➔ Breakdown by IP class

```
SELECT timeslot, totalRequests, cached, uncached, requestsEncrypted, requestsUnencrypted, totalBytes, bytesEncrypted, bytesUnencrypted, threats, pageViews, responseStatus,
    requestsByResponseStatus, contentType, requestsByContentType, bytesByContentType, country, requestsByCountry, bytesByCountry, threatsByCountry, ipClass, requestsByIPClass,
    browser, pageViewsByBrowser FROM (SELECT timeslot, totalRequests, cached, uncached, requestsEncrypted, requestsUnencrypted, totalBytes, bytesEncrypted, bytesUnencrypted,
    threats, pageViews, responseStatus, requestsByResponseStatus, contentType, requestsByContentType, bytesByContentType, country, requestsByCountry, bytesByCountry,
    threatsByCountry, ipClass, requestsByIPClass FROM (SELECT timeslot, totalRequests, cached, uncached, requestsEncrypted, requestsUnencrypted, totalBytes, bytesEncrypted,
    bytesUnencrypted, threats, pageViews, responseStatus, requestsByResponseStatus, contentType, requestsByContentType, bytesByContentType, country, requestsByCountry,
    bytesByCountry, threatsByCountry FROM (SELECT timeslot, totalRequests, cached, uncached, requestsEncrypted, requestsUnencrypted, totalBytes, bytesEncrypted, bytesUnencrypted,
    threats, pageViews, responseStatus, requestsByResponseStatus, contentType, requestsByContentType, bytesByContentType FROM (SELECT timeslot, totalRequests, cached, uncached,
    requestsEncrypted, requestsUnencrypted, totalBytes, bytesEncrypted, bytesUnencrypted, threats, pageViews, responseStatus, requestsByResponseStatus FROM (SELECT timeslot, sum(
    requests) AS totalRequests, sum(cacheable) AS cached, sum(cacheMiss) AS uncached, sumIf(requests, clientSSLProtocol NOT IN ('unknown', 'none')) AS requestsEncrypted, sumIf(
    requests, clientSSLProtocol IN ('unknown', 'none')) AS requestsUnencrypted, sum(bytes) AS totalBytes, sumIf(bytes, clientSSLProtocol NOT IN ('unknown', 'none')) AS
    bytesEncrypted, sumIf(bytes, clientSSLProtocol IN ('unknown', 'none')) AS bytesUnencrypted, sumIf(requests, (edgePathingOp IN ('chl', 'ban')) AND (edgePathingStatus NOT IN (8, 9
    , 22, 23))) AS threats, sumIf(requests, (edgeResponseStatus = 200) AND (edgeResponseContentType = 691)) AS pageViews FROM requests_1m_breakdown WHERE (zoneId = 4240) AND ((
    timeslot >= '2017-08-09 00:00:00') AND (timeslot <= '2017-08-09 00:30:00')) AND (date = '2017-08-09') GROUP BY timeslot ORDER BY timeslot ASC ) ANY LEFT JOIN (SELECT timeslot,
    groupArray(edgeResponseStatus) AS responseStatus, groupArray(requests) AS requestsByResponseStatus FROM (SELECT timeslot, edgeResponseStatus, sum(requests) AS requests FROM
    requests_1m_breakdown WHERE (zoneId = 4240) AND ((timeslot >= '2017-08-09 00:00:00') AND (timeslot <= '2017-08-09 00:30:00')) AND (date = '2017-08-09') GROUP BY timeslot,
    edgeResponseStatus ) GROUP BY timeslot ORDER BY timeslot ASC ) USING (timeslot) ) ANY LEFT JOIN (SELECT timeslot, groupArray(contentType) AS contentType, groupArray(requests) AS
     requestsByContentType, groupArray(bytes) AS bytesByContentType FROM (SELECT timeslot, dictGetString('mime', 'name', toUInt64(edgeResponseContentType)) AS contentType, sum(
    requests) AS requests, sum(bytes) AS bytes FROM requests_1m_breakdown WHERE (zoneId = 4240) AND ((timeslot >= '2017-08-09 00:00:00') AND (timeslot <= '2017-08-09 00:30:00')) AND
     (date = '2017-08-09') GROUP BY timeslot, edgeResponseContentType ) GROUP BY timeslot ORDER BY timeslot ASC ) USING (timeslot) ) ANY LEFT JOIN (SELECT timeslot, groupArray(
    country) AS country, groupArray(totalRequests) AS requestsByCountry, groupArray(bytes) AS bytesByCountry, groupArray(threats) AS threatsByCountry FROM (SELECT timeslot,
    dictGetString('country', 'alpha2', toUInt64(clientCountry)) AS country, sum(requests) AS totalRequests, sum(bytes) AS bytes, sumIf(requests, (edgePathingOp IN ('chl', 'ban'))
    AND (edgePathingStatus NOT IN (8, 9, 22, 23))) AS threats FROM requests_1m_breakdown WHERE (zoneId = 4240) AND ((timeslot >= '2017-08-09 00:00:00') AND ((timeslot <= '2017-08-09
    00:30:00')) AND (date = '2017-08-09') GROUP BY timeslot, clientCountry ) GROUP BY timeslot ORDER BY timeslot ASC ) USING (timeslot) ) ANY LEFT JOIN (SELECT timeslot, groupArray(
    toString(clientIPClass)) AS ipClass, groupArray(requests) AS requestsByIPClass FROM (SELECT timeslot, clientIPClass, sum(requests) AS requests FROM requests_1m_breakdown WHERE (
    zoneId = 4240) AND ((timeslot >= '2017-08-09 00:00:00') AND (timeslot <= '2017-08-09 00:30:00')) AND (date = '2017-08-09') GROUP BY timeslot, clientIPClass ) GROUP BY timeslot
    ORDER BY timeslot ASC ) USING (timeslot) ) ANY LEFT JOIN (SELECT timeslot, groupArray(browser) AS browser, groupArray(pageViews) AS pageViewsByBrowser FROM (SELECT timeslot,
    dictGetString('browser', 'name', toUInt64(uaBrowserFamily)) AS browser, sumIf(requests, (edgeResponseStatus = 200) AND (edgeResponseContentType = 691)) AS pageViews FROM
    requests_1m_breakdown WHERE (zoneId = 4240) AND ((timeslot >= '2017-08-09 00:00:00') AND (timeslot <= '2017-08-09 00:30:00')) AND (date = '2017-08-09') GROUP BY timeslot,
    uaBrowserFamily ) GROUP BY timeslot ORDER BY timeslot ASC ) USING (timeslot)
```

CLOUDFLARE

Aggregations schema design #2

Use just 2 tables:

- ➔ **requests_1m** (SummingMergeTree engine)
  - ◆ Stores minutely breakdowns: counters and “maps”
  - ◆ Using sum / sumMap to finalize aggregation
- ➔ **requests_1m_uniques** (AggregatingMergeTree engine)
  - ◆ Stores unique IPv4 / IPv6 states
  - ◆ Using uniqMerge to finalize aggregation
- ➔ In the future: add states merge into SummingMergeTree

sumMap(key, value)

Totals the 'value' array according to the keys specified in the 'key' array. The number of elements in 'key' and 'value' must be the same for each row that is totaled. Returns a tuple of two arrays: keys in sorted order, and values summed for the corresponding keys.

Example:

```
CREATE TABLE sum_map(
    date Date,
    timeslot DateTime,
    statusMap Nested(
        status UInt16,
        requests UInt64
    )
) ENGINE = Log;
INSERT INTO sum_map VALUES
    ('2000-01-01', '2000-01-01 00:00:00', [1, 2, 3], [10, 10, 10]),
    ('2000-01-01', '2000-01-01 00:00:00', [3, 4, 5], [10, 10, 10]),
    ('2000-01-01', '2000-01-01 00:01:00', [4, 5, 6], [10, 10, 10]),
    ('2000-01-01', '2000-01-01 00:01:00', [6, 7, 8], [10, 10, 10]);
SELECT
    timeslot,
    sumMap(statusMap.status, statusMap.requests)
FROM sum_map
GROUP BY timeslot
```

```
┌────────────timeslot─┬─sumMap(statusMap.status, statusMap.requests)─┐
│ 2000-01-01 00:00:00 │ ([1,2,3,4,5],[10,10,20,10,10])               │
│ 2000-01-01 00:01:00 │ ([4,5,6,7,8],[10,10,20,10,10])               │
└─────────────────────┴──────────────────────────────────────────────┘
```

CLOUDFLARE

Aggregations schema design #2

Citus

```
dw=# \d fdw_zoneagg.zone_request_1mm
                Foreign table "fdw_zoneagg.zone_request_1mm"
          Column           |           Type           | Modifiers |
---------------------------+--------------------------+-----------+
 t                         | timestamp with time zone |           |
 zoneid                    | bigint                   |           |
 partid                    | smallint                 |           |
 hosterid                  | bigint                   |           |
 ip_version                | smallint                 |           |
 requests                  | bigint                   |           |
 pageviews                 | bigint                   |           |
 bytes_client              | bigint                   |           |
 bytes_origin              | bigint                   |           |
 uniques                   | hll                      |           |
 requests_by_http_status   | hstore                   |           |
 requests_by_ip_type       | hstore                   |           |
 requests_by_colo          | hstore                   |           |
 requests_by_content_type  | hstore                   |           |
 requests_by_country       | hstore                   |           |
 requests_by_useragent     | hstore                   |           |
 bytes_by_colo             | hstore                   |           |
 bytes_by_country          | hstore                   |           |
 bytes_by_ip_type          | hstore                   |           |
 bytes_by_content_type     | hstore                   |           |
 bytes_by_useragent        | hstore                   |           |
 pageviews_by_ip_type      | hstore                   |           |
 pageviews_by_useragent    | hstore                   |           |
 requests_origin           | bigint                   |           |
 threats_by_country        | hstore                   |           |
 threats_by_pathing        | hstore                   |           |
 zone_plan                 | character(3)             |           |
 requests_by_ssl           | hstore                   |           |
 bytes_by_ssl              | hstore                   |           |
Server: zoneagg
```

ClickHouse

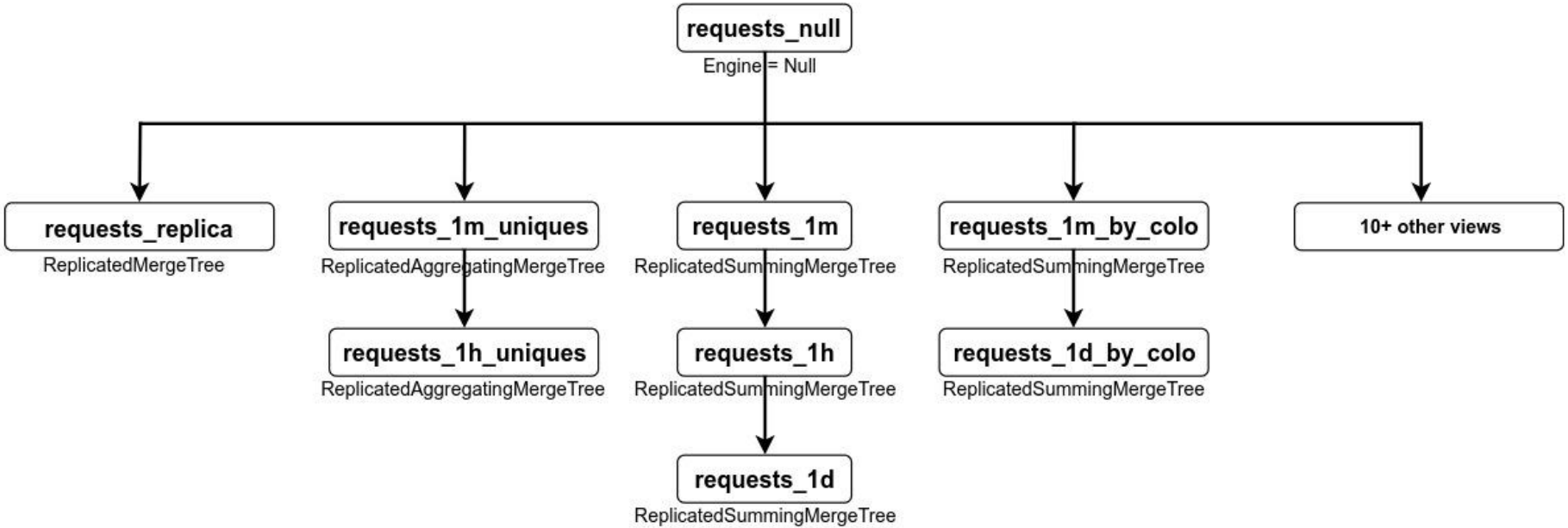
```
:) desc requests_1m;

DESCRIBE TABLE requests_1m

┌─name──────────────────────────────────┬─type──────────┐
│ date                                  │ Date          │
│ zoneId                                │ UInt32        │
│ timeslot                              │ DateTime      │
│ requests                              │ UInt64        │
│ cachedRequests                        │ UInt64        │
│ encryptedRequests                     │ UInt64        │
│ bytes                                 │ UInt64        │
│ cachedBytes                           │ UInt64        │
│ encryptedBytes                        │ UInt64        │
│ pageViews                             │ UInt64        │
│ threats                               │ UInt64        │
│ responseStatusMap.edgeResponseStatus  │ Array(UInt16) │
│ responseStatusMap.requests            │ Array(UInt64) │
│ contentTypeMap.edgeResponseContentType│ Array(UInt32) │
│ contentTypeMap.requests               │ Array(UInt64) │
│ contentTypeMap.bytes                  │ Array(UInt64) │
│ countryMap.clientCountry              │ Array(UInt16) │
│ countryMap.requests                   │ Array(UInt64) │
│ countryMap.bytes                      │ Array(UInt64) │
│ countryMap.threats                    │ Array(UInt64) │
│ ipClassMap.clientIPClass              │ Array(UInt8)  │
│ ipClassMap.requests                   │ Array(UInt64) │
│ threatPathingMap.threatPathingId      │ Array(UInt32) │
│ threatPathingMap.requests             │ Array(UInt64) │
│ browserMap.uaBrowserFamily            │ Array(UInt8)  │
│ browserMap.pageViews                  │ Array(UInt64) │
```

CLOUDFLARE

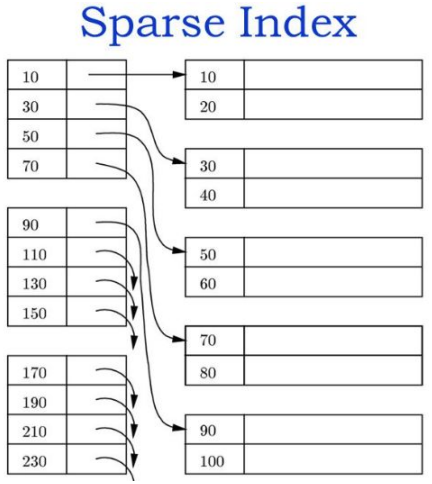
Final schemas design



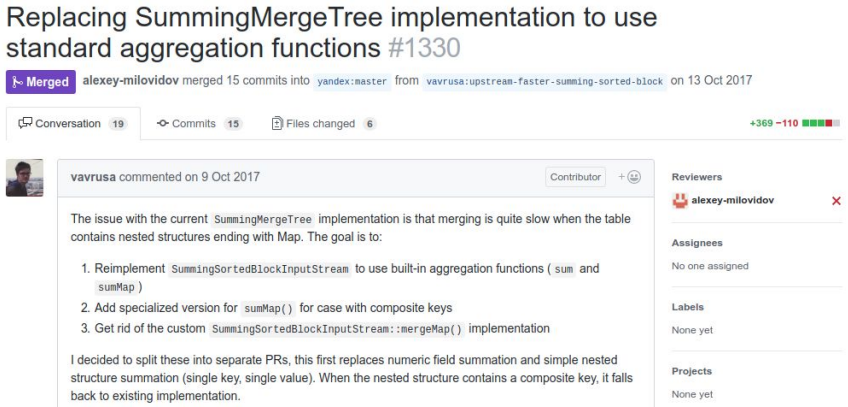
CLOUDFLARE

Tuning ClickHouse performance

- ➔ Index granularity choice
  - Default – 8192
  - Non-aggregated requests – changed to 16384
  - Aggregations tables – changed to 32
  - Query latency 1/2
  - Throughput x3
  - Rule of thumb: match index granularity to the number of rows scanned per query

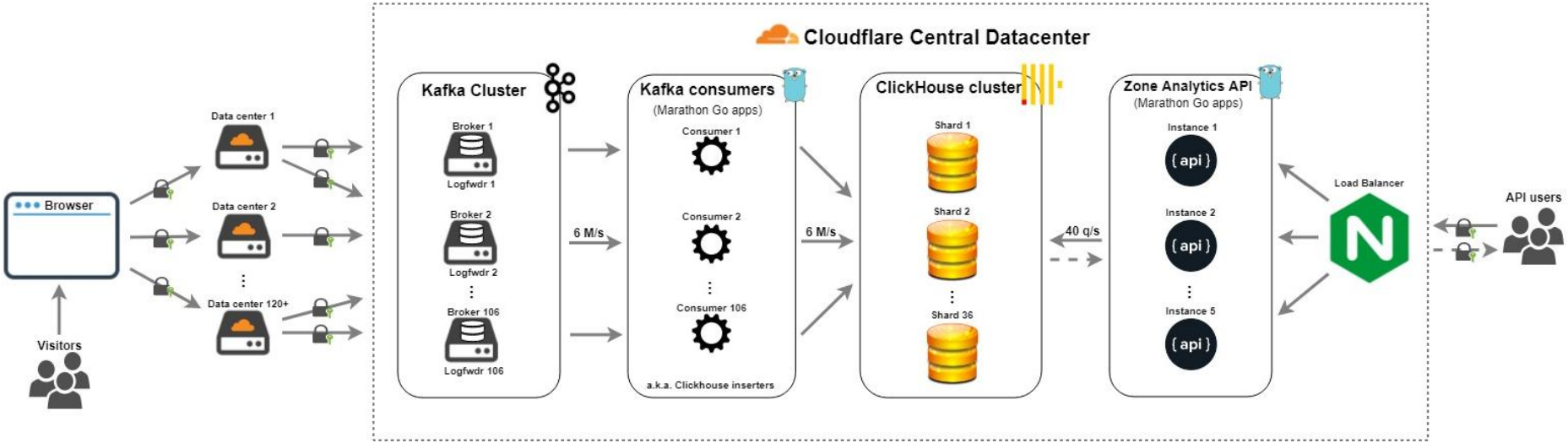


- ➔ ClickHouse optimizations
  - SummingMergeTree optimizations
  - Maps merge speed x7!



CLOUDFLARE

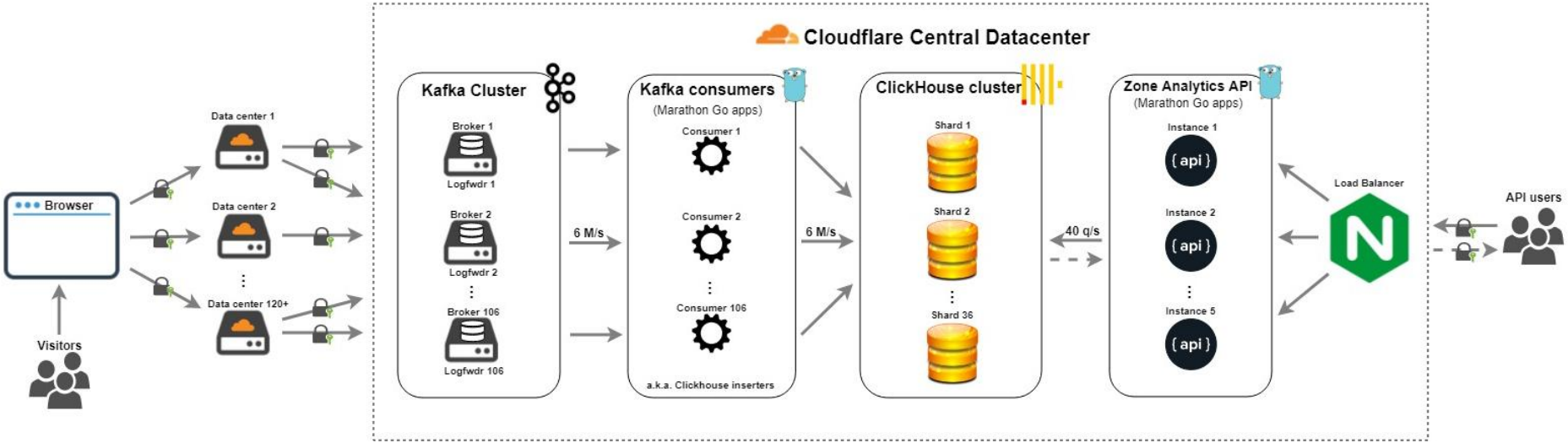
New pipeline



- ➔ Log Forwarder
- ➔ Kafka cluster
- ➔ Kafka consumers*
- ➔ ~~Postgres database~~
- ➔ ~~Citus Cluster~~
- ➔ Zone Analytics API*
- ➔ ~~PHP API~~
- ➔ Load Balancer

CLOUDFLARE

New pipeline advantages



- ➔ No SPoF
- ➔ Fault-tolerant
- ➔ Scalable
- ➔ Reduced complexity
- ➔ Improved API throughput and latency
- ➔ Easier to operate
- ➔ Decreased amount of incidents

CLOUDFLARE

Our ClickHouse cluster

36	x3	12M+	50Gbit+	4PB+
Nodes	Replication factor	Row Insertion/s	Insertion Throughput/s	Raid-0 Spinning Disks

Each node:

- **CPU** - 40 logical cores E5-2630 v3 @ 2.40 GHz
- **RAM** - 256 GB RAM
- **Disks** - 12 x 10 TB Seagate ST10000NM0016-1TT101
- **Network** - 2 x 25G Mellanox ConnectX-4

CLOUDFLARE

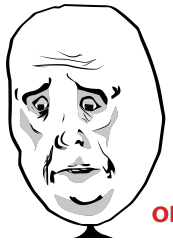
Cloudflare vs Bloomberg

36	x3	12M+	50Gbit+	4PB+
Nodes	Replication factor	Row Insertion/s	Insertion Throughput/s	Raid-0 Spinning Disks

Each node:

- **CPU** - 40 logical cores E5-2630 v3 @ 2.40 GHz
- **RAM** - 256 GB RAM
- **Disks** - 12 x 10 TB Seagate ST10000NM0016-1TT101
- **Network** - 2 x 25G Mellanox ConnectX-4

102	x3	60M+	80Gbit+	1PB+
Nodes	Replication factor	Row Insertion/s	Insertion Throughput/s 42 fields of netflow data	NVMe SSDs



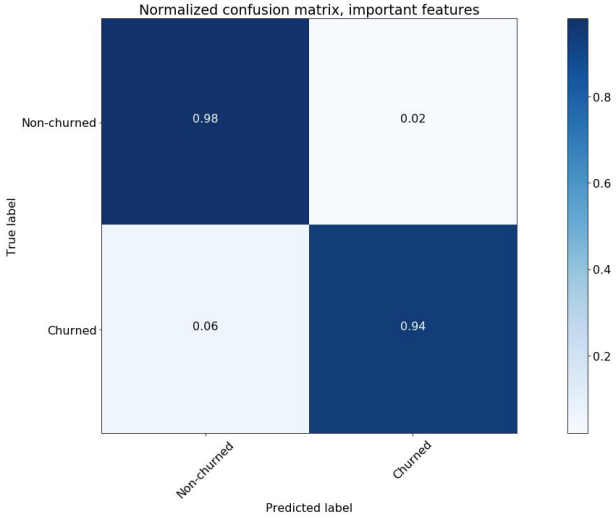
CLOUDFLARE

What we're working on

- ➔ LogShare prototype (currently on HDFS)
- ➔ Usage based billing products (currently on Flink + Citus)
- ➔ Log Push
- ➔ Logs SQL API
- ➔ Churn prediction with CatBoost

SQL

CatBoost



CLOUDFLARE

Thanks!

Questions?

- ➔ Detailed blog post: cflare.co/6m-reqs
- ➔ Twitter: twitter.com/b0ch4r0v
- ➔ Github: github.com/bocharov

CLOUDFLARE