Altinity GUEST WEBINAR

BIG DATA AND BEAUTIFUL VIDEO:

How ClickHouse enables Mux to Deliver Content at Scale

MUX

Presenter Bios

Adam Brown -- Head of Technology and Architecture

Co-founder of Mux with extensive experience in video encoding and delivery going back to Zencoder

Robert Hodges - Altinity CEO

30+ years on DBMS plus virtualization and security. ClickHouse is DBMS #20

Company Intros

mux.com

Mux Video is an API-first platform, powered by data and designed by video experts to make beautiful video possible for every development team.

www.altinity.com

Leading software and services provider for ClickHouse

Major committer and community sponsor in US and Western Europe

MUX

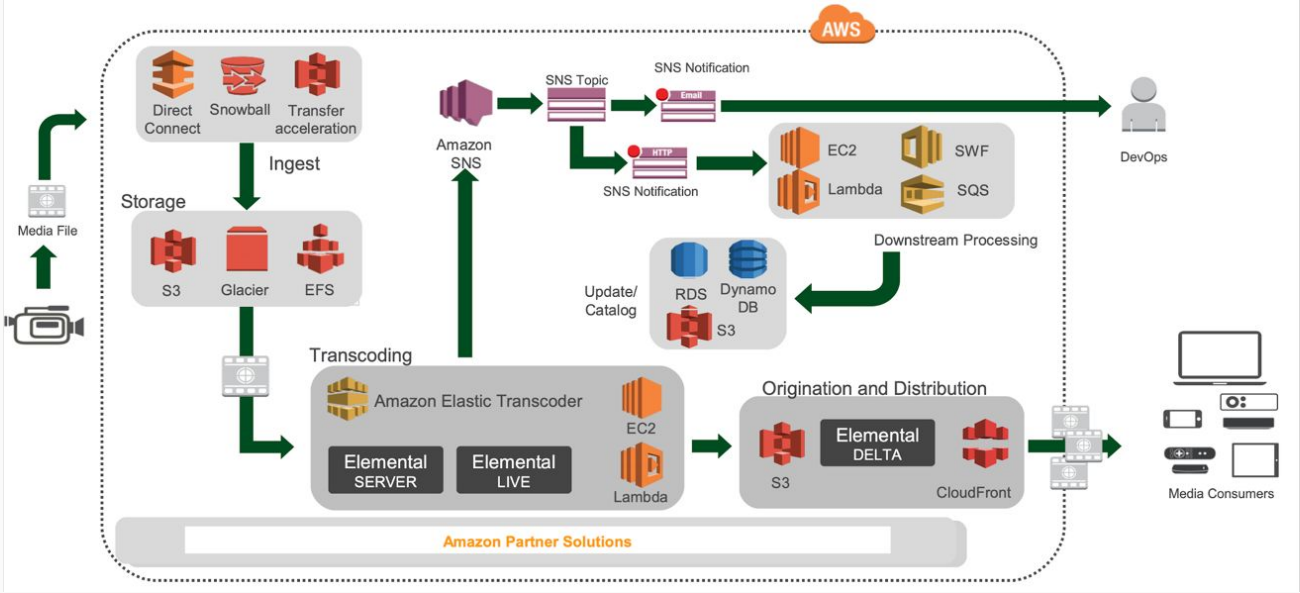
Data and Video Delivery

Altinity

Standard Media Workflow (w/o Mux)

Any Video In

Upload
Live Stream



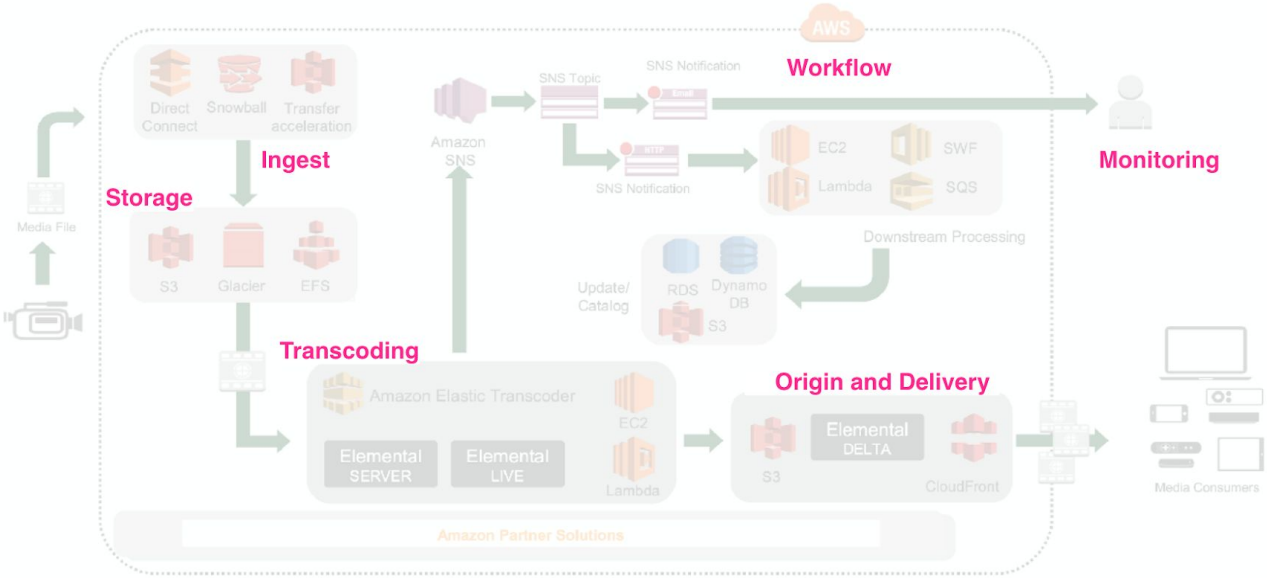
Any Device

Desktop
Mobile
TV

Standard Media Workflow

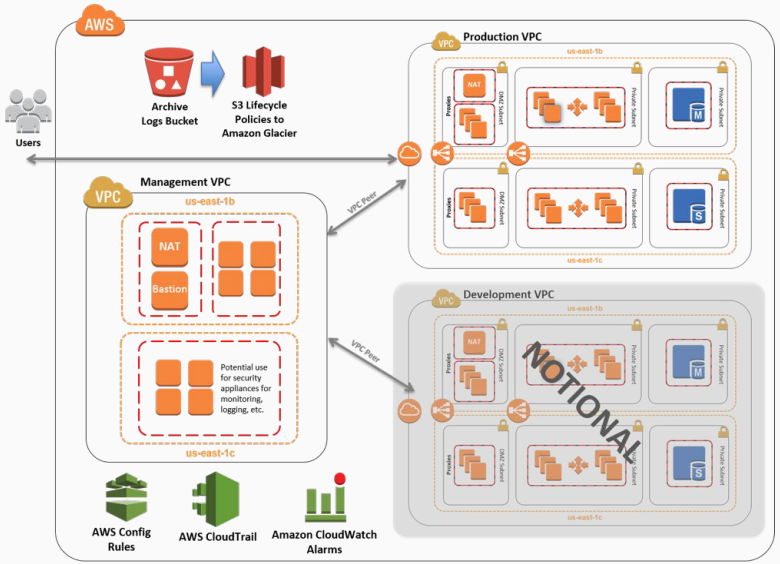
Any Video In

Upload
Live Stream

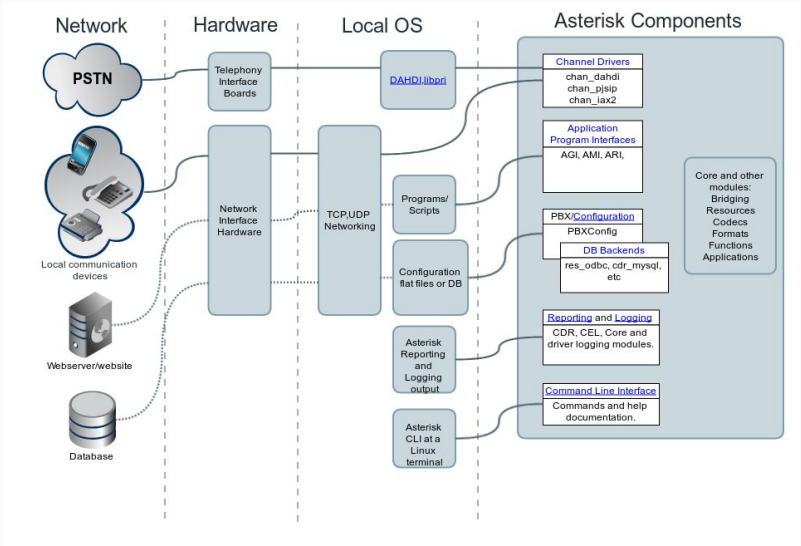


Any Device

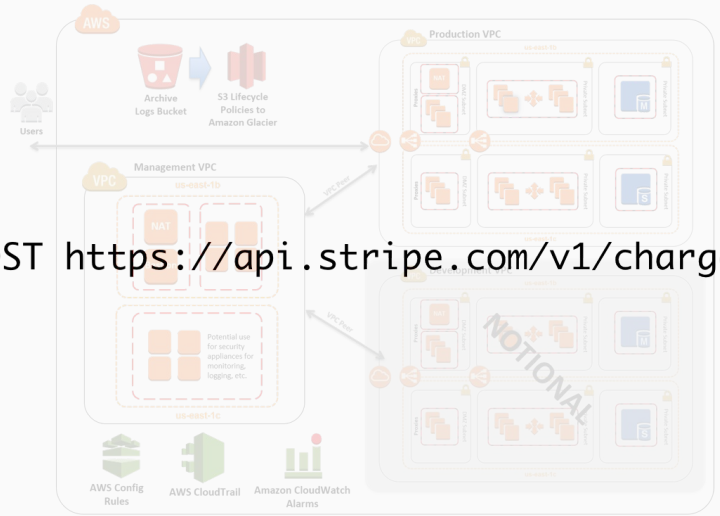
Desktop
Mobile
TV



Payments before Stripe

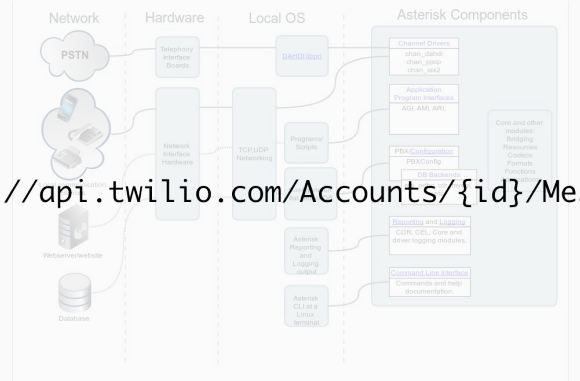


Messaging before Twilio



POST https://api.stripe.com/v1/charges

Payments using Stripe



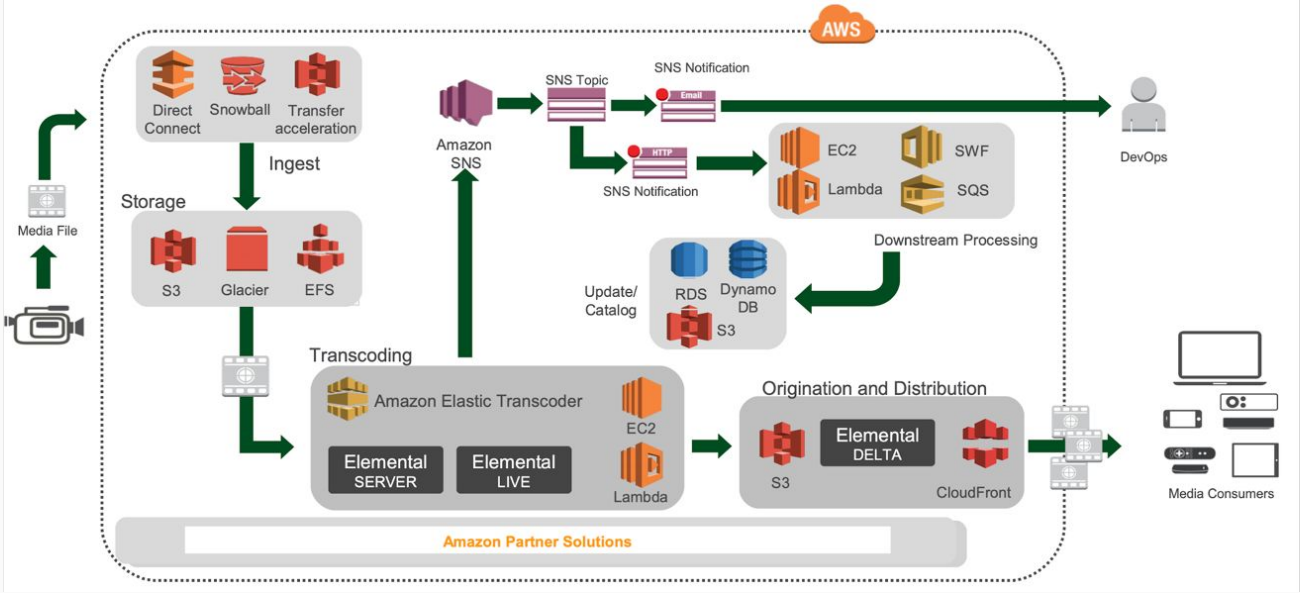
POST https://api.twilio.com/Accounts/{id}/Messages.json

Messaging using Twilio

Standard Media Workflow (w/o Mux)

Any Video In

Upload
Live Stream



Any Device

Desktop
Mobile
TV

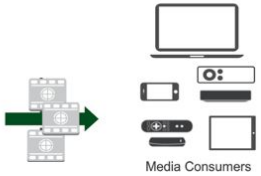
Any Video In

Upload
Live Stream


```
POST https://api.mux.com/video/assets

{
  "input": "https://storage.googleapis.com/muxdemofiles/mux-video-intro.mp4",
  "playback_policy": "public"
}
```

MUX

https://stream.mux.com/02fhbf00FwC1V2HF4wTyMqq8JVFMQvuU00e.m3u8

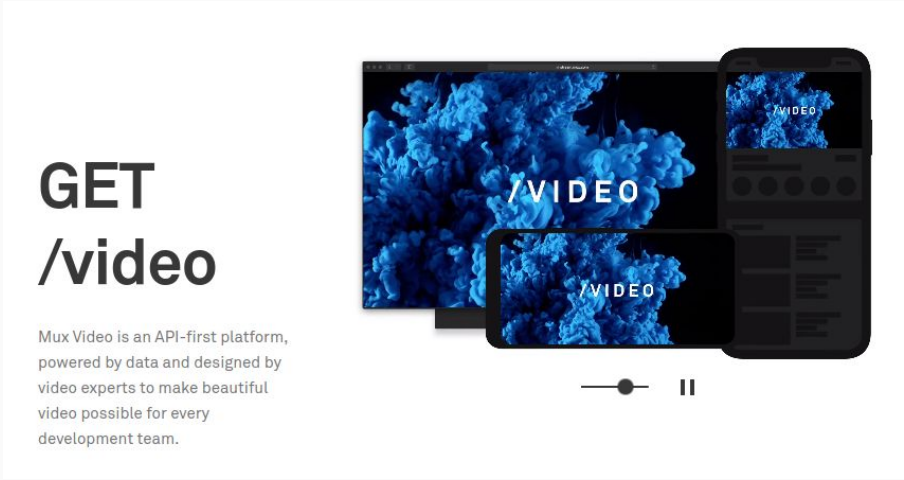


Any Device

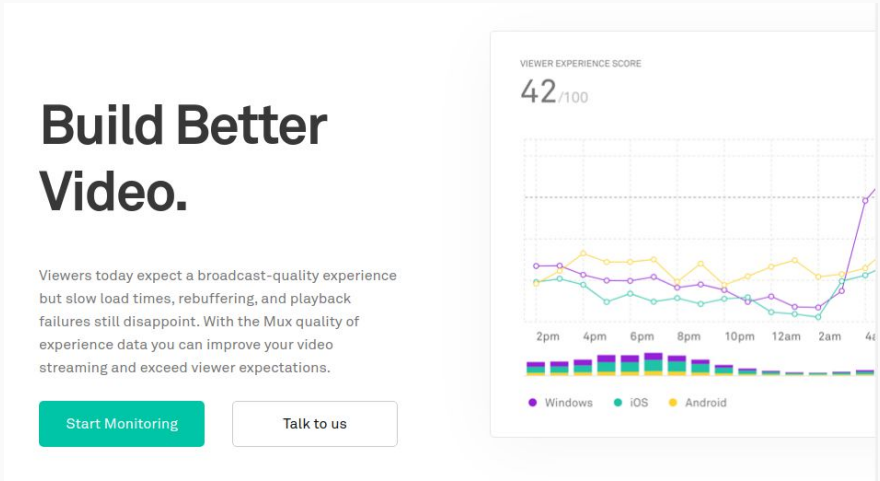
Desktop
Mobile
TV

Mux Products

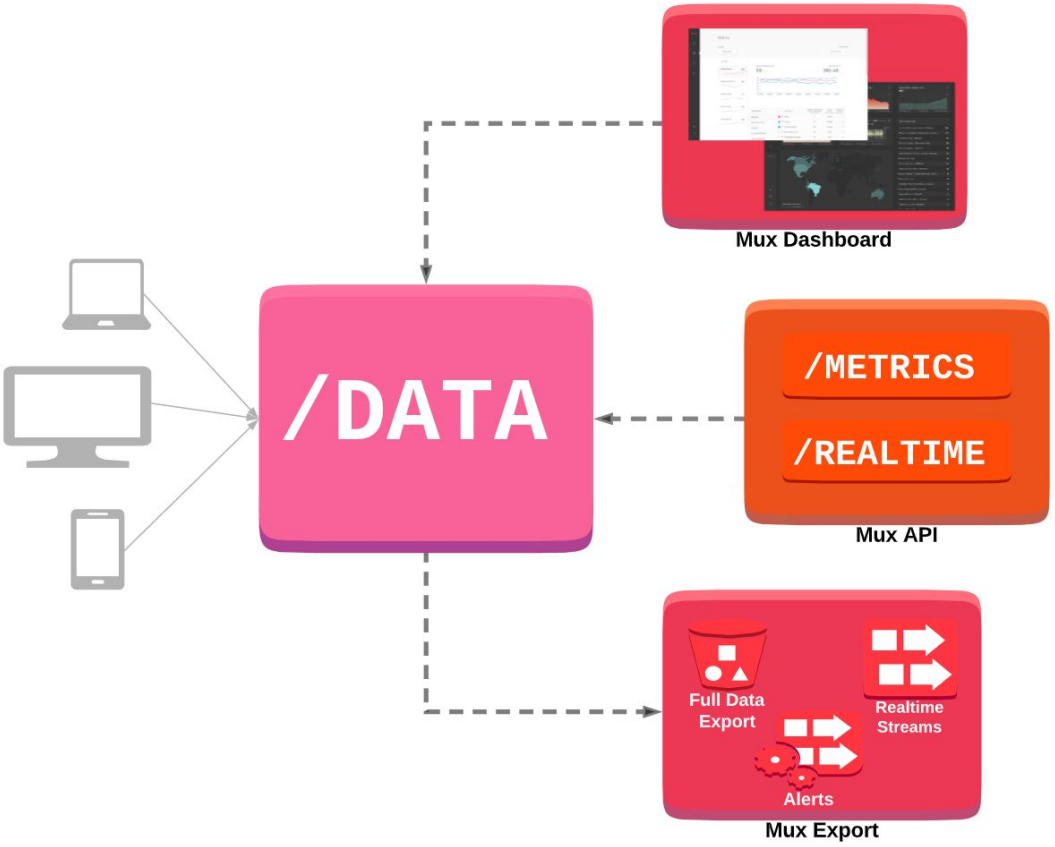
Mux Video



Mux Data



Mux Data Overview



Data Sources

- Mux Data SDKS
- CDN Logs
- Internal Monitoring

SDKs

Web	Mobile
Generic HTML5	Apple iOS (8+)
Video.js	Android/ExoPlayer (v2)
Hls.js	Android MediaPlayer
Dash.js	Brightcove Player SDK for Android
Shaka Player	Brightcove Player SDK for iOS
Brightcove (5.x and 6.x)	**OTT**
Azure Media Player	Roku
JW Player (7 and 8)	Chromecast
Bitmovin Player (5.x, 6.x, and 7.x)	Apple TV tvOS (9+)
Ooyala Player (V4)	Fire TV
THEOplayer (2.x)	Samsung Tizen
Flowplayer (7.x)	LG WebOS
Comcast Technology Solutions/thePlatform PDK (5 & 6)	
Akamai Media Player	

Example Use Case

Video delivered by CDN for AS701 and AS22394

StackPath

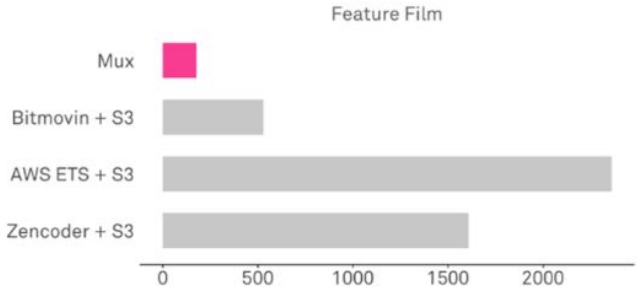
fastly

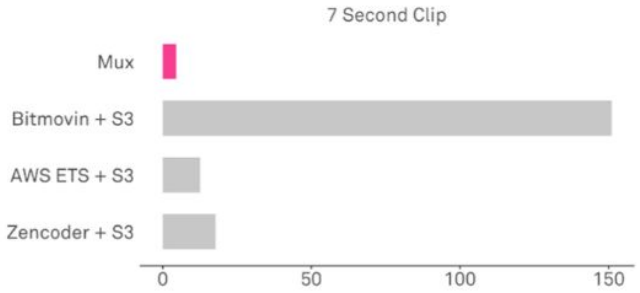
amazon cloudfront

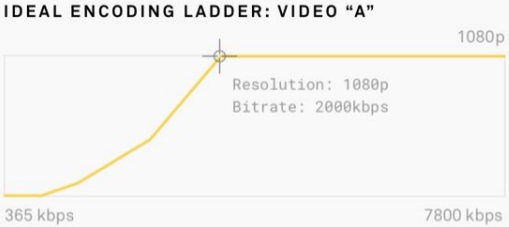
Altinity

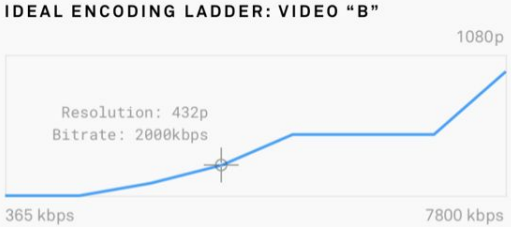
MUX

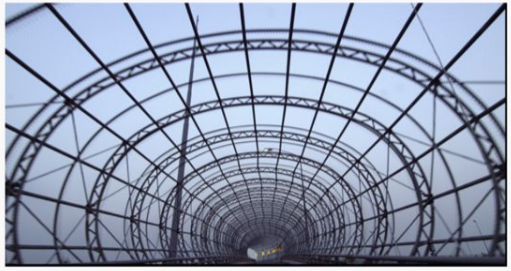
Data drives video engine











Altinity

MUX

MUX

ClickHouse features that enable Mux

Altinity

Introducing the MergeTree table engine

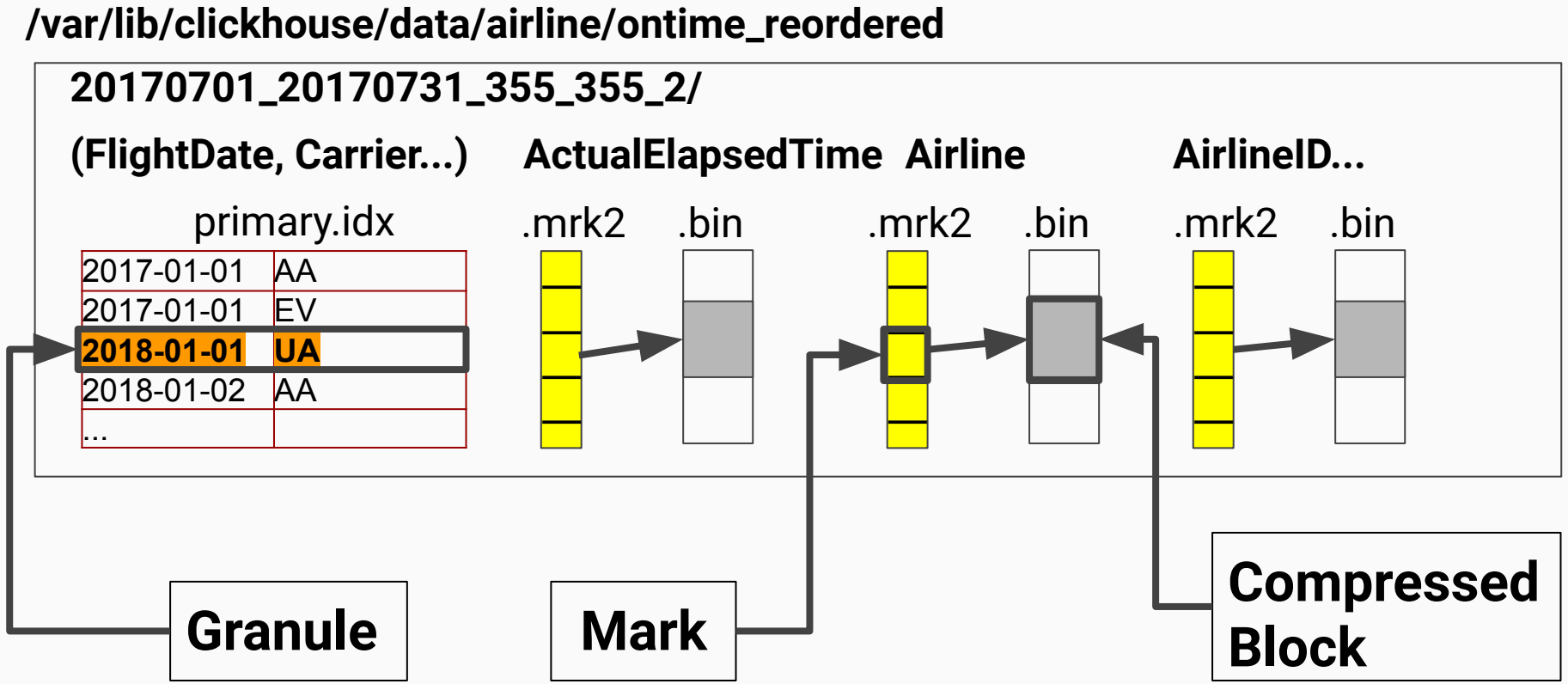
```
CREATE TABLE ontime (
   Year UInt16,
   Quarter UInt8,
   Month UInt8,
   ...
) ENGINE = MergeTree()
PARTITION BY toYYYYMM(FlightDate)
ORDER BY (Carrier, FlightDate)
```

Table engine type

How to break data into parts

How to index and sort data in each part

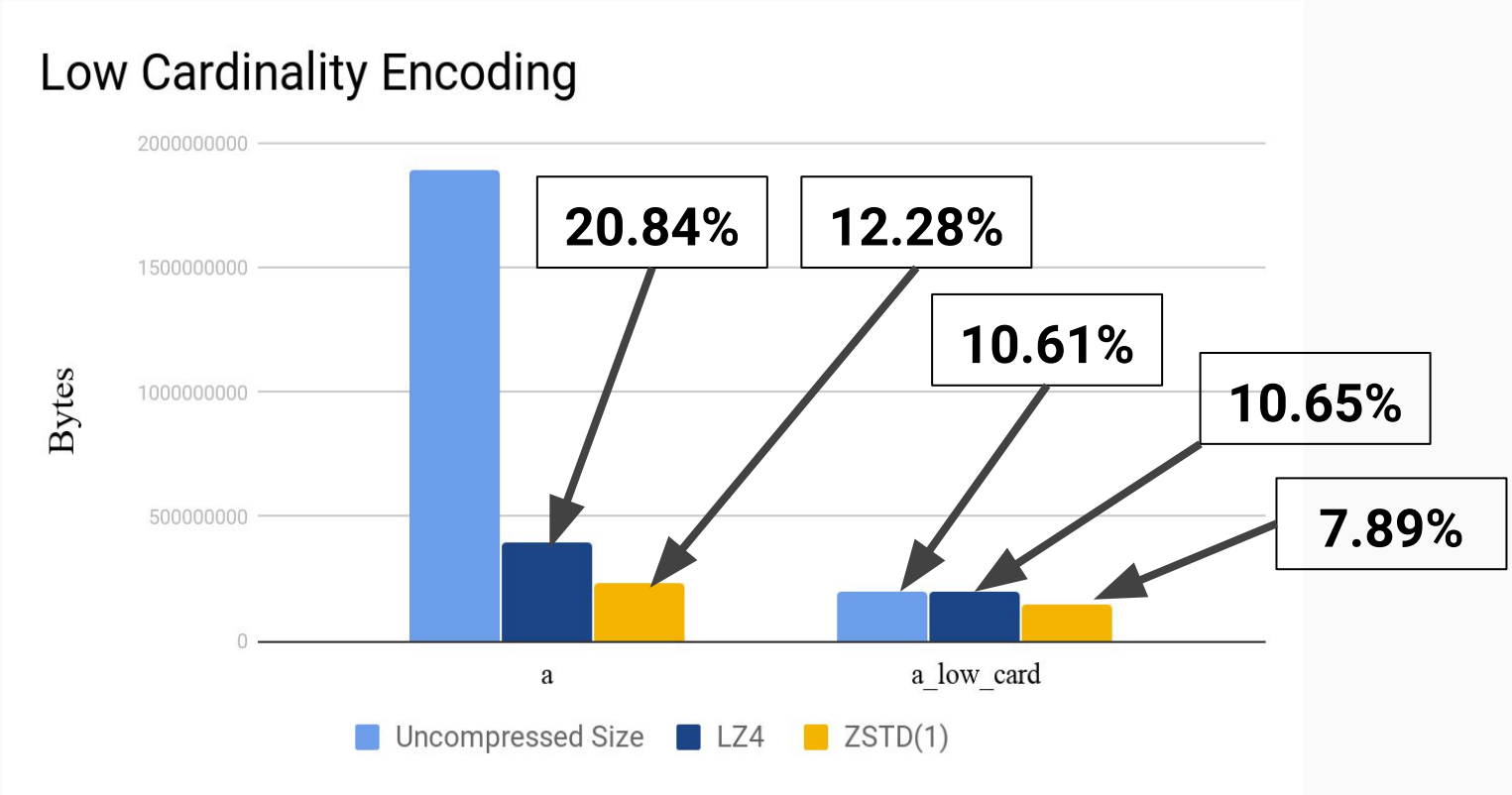
MergeTree layout within a single part



Compression and codecs are configurable

```
CREATE TABLE test_codecs (
  a_lz4 String CODEC(LZ4),
  a_zstd String DEFAULT a_lz4 CODEC(ZSTD),
  a_lc_lz4 LowCardinality(String) DEFAULT a_lz4 CODEC(LZ4),
  a_lc_zstd LowCardinality(String) DEFAULT a_lz4 CODEC(ZSTD)
)
Engine = MergeTree
PARTITION BY tuple() ORDER BY tuple();
```

Effect on storage size is dramatic

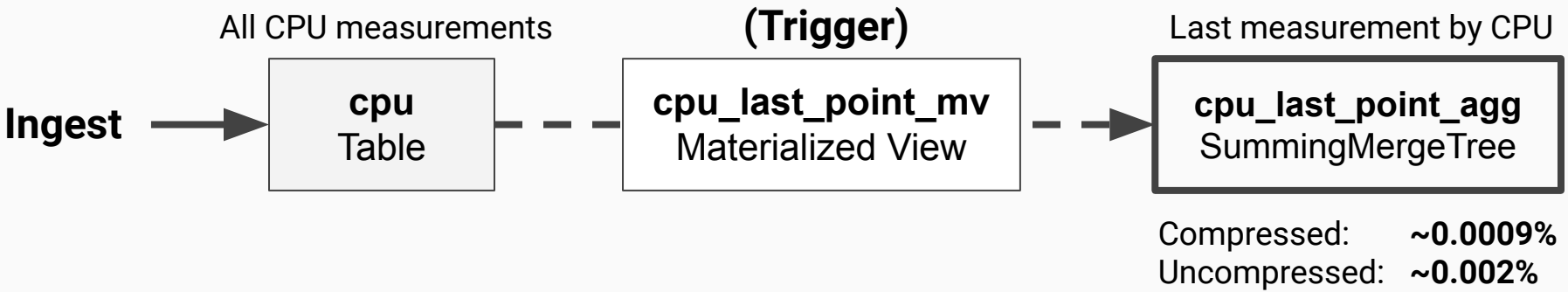


How do you get those nice numbers?

```
SELECT name AS col,
    sum(data_uncompressed_bytes) AS uncompressed,
    sum(data_compressed_bytes) AS compressed,
    round((compressed / uncompressed) * 100., 2) AS pct,
    bar(pct, 0., 100., 20) AS bar_graph
FROM system.columns
WHERE (database = currentDatabase()) AND (table = 'test_codecs')
GROUP BY name ORDER BY pct DESC, name ASC
```

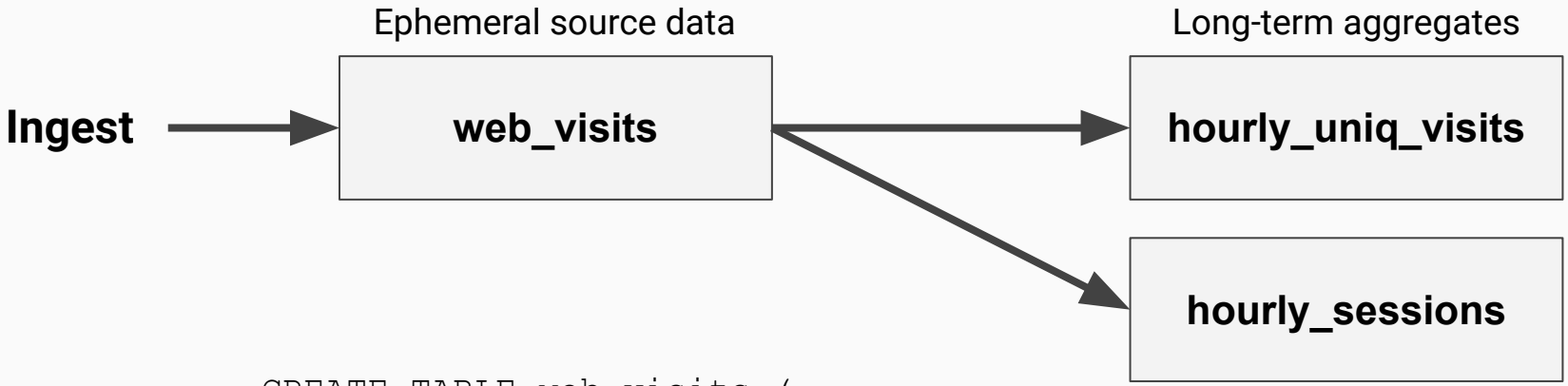
col	uncompressed	compressed	pct	bar_graph
a_lc_lz4	200446439	201154656	100.35	
a_lc_zstd	200446439	148996404	74.33	
a_lz4	1889003550	393713202	20.84	
a_zstd	1889003550	231975292	12.28	

Materialized views restructure/reduce data




```
CREATE MATERIALIZED VIEW cpu_last_point_mv
TO cpu_last_point_agg
AS SELECT
  cpu_id,
  maxState(created_at) AS max_created_at,
  . . .
  argMaxState(usage_idle, created_at) AS usage_idle
FROM cpu GROUP BY cpu_id
```

Pattern: TTLs + downsampled views




```
CREATE TABLE web_visits (
  time DateTime CODEC(DoubleDelta,
ZSTD),
  session_id UInt64,
  visitor_id UInt64, . . .
) Engine = MergeTree
PARTITION BY toDate(time)
ORDER BY (session id, time)
TTL time + INTERVAL 7 DAY
```

Skip indexes cut down on I/O

```
SET allow_experimental_data_skipping_indices=1;

ALTER TABLE ontime ADD INDEX
  dest_name Dest TYPE ngrambf_v1(3, 512, 2, 0) GRANULARITY 1

ALTER TABLE ontime ADD INDEX
  cname Carrier TYPE set(0) GRANULARITY 1

OPTIMIZE TABLE ontime FINAL
-- OR, in current releases
ALTER TABLE ontime
  UPDATE Dest=Dest, Carrier=Carrier
    WHERE 1=1
```

Effectiveness depends on data distribution

```
SELECT
  Year, count(*) AS flights,
  sum(Cancelled) / flights AS cancelled,
  sum(DepDel15) / flights AS delayed_15
FROM airline.ontime WHERE [Column] = [Value] GROUP BY Year
```

Column	Value	Index	Count	Rows Processed	Query Response
Dest	PPG	ngrambf_v1	525	4.30M	**0.053**
Dest	ATL	ngrambf_v1	9,360,581	166.81M	0.622
Carrier	ML	set	70,622	3.39M	**0.090**
Carrier	WN	set	25,918,402	166.24M	0.566

More table engines: CollapsingMergeTree

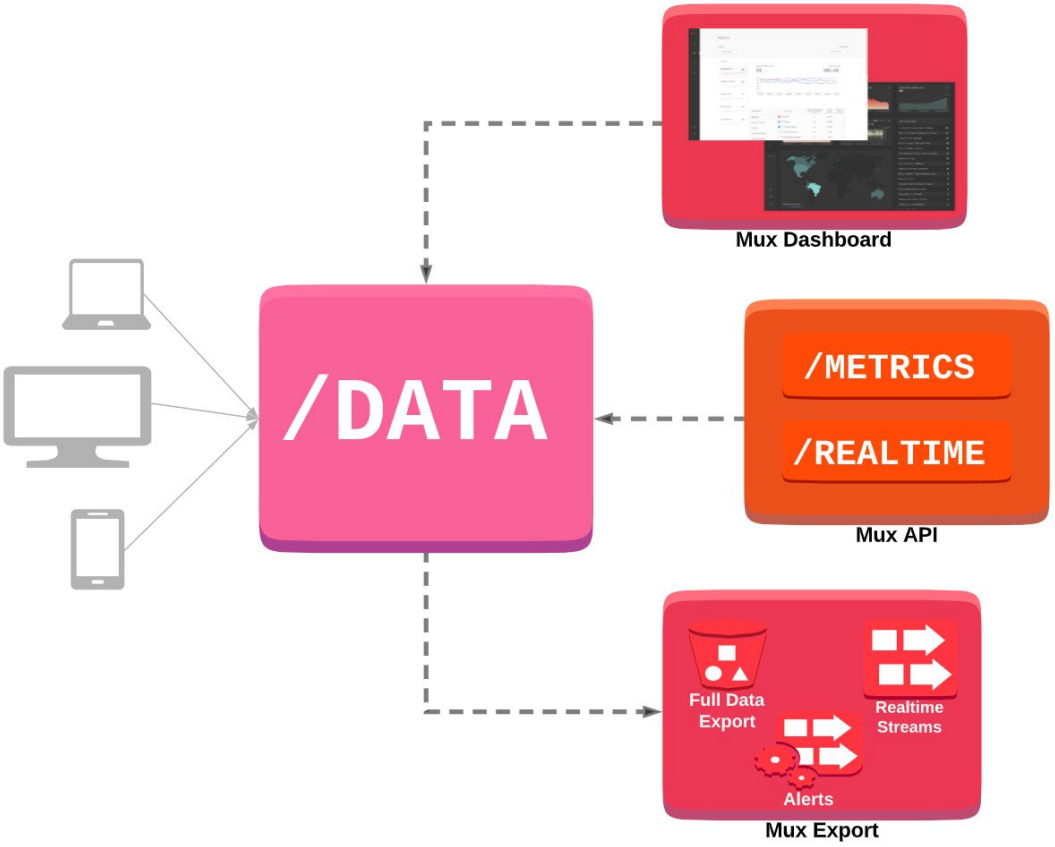
```
CREATE TABLE collapse (user_id UInt64, views UInt64, sign Int8)
ENGINE = CollapsingMergeTree(sign)
PARTITION BY tuple() ORDER BY user_id;

INSERT INTO collapse VALUES (32, 55, 1);
INSERT INTO collapse VALUES (32, 55, -1);
INSERT INTO collapse VALUES (32, 98, 1);

SELECT * FROM collapse FINAL
┌─user_id─┬─views─┬─sign─┐
│      32 │    98 │    1 │
└─────────┴───────┴──────┘
```

MUX

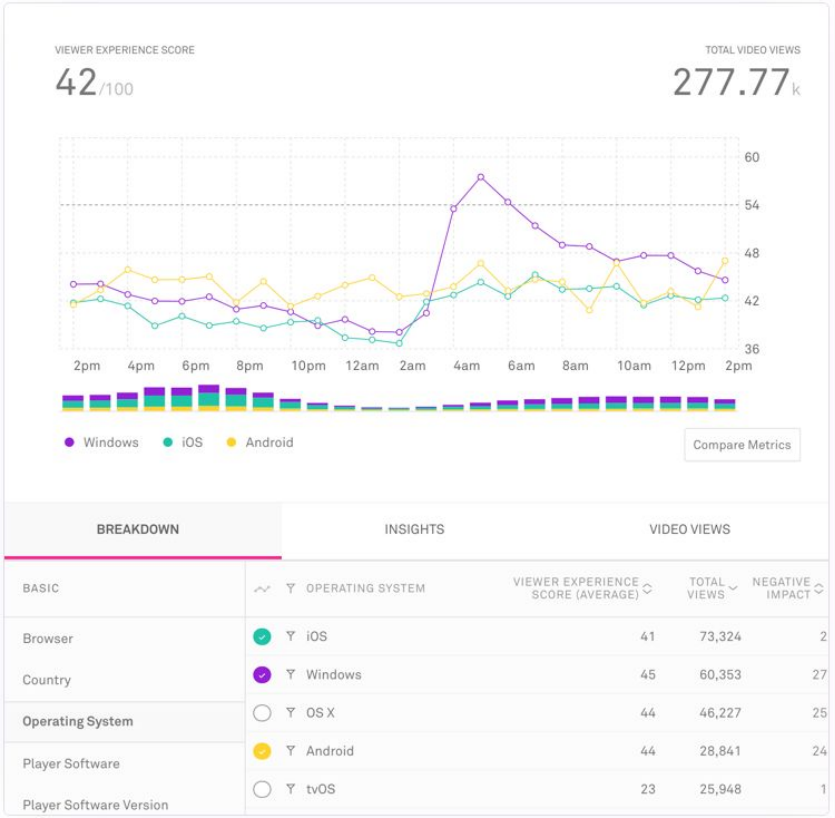
Mux Experience with ClickHouse

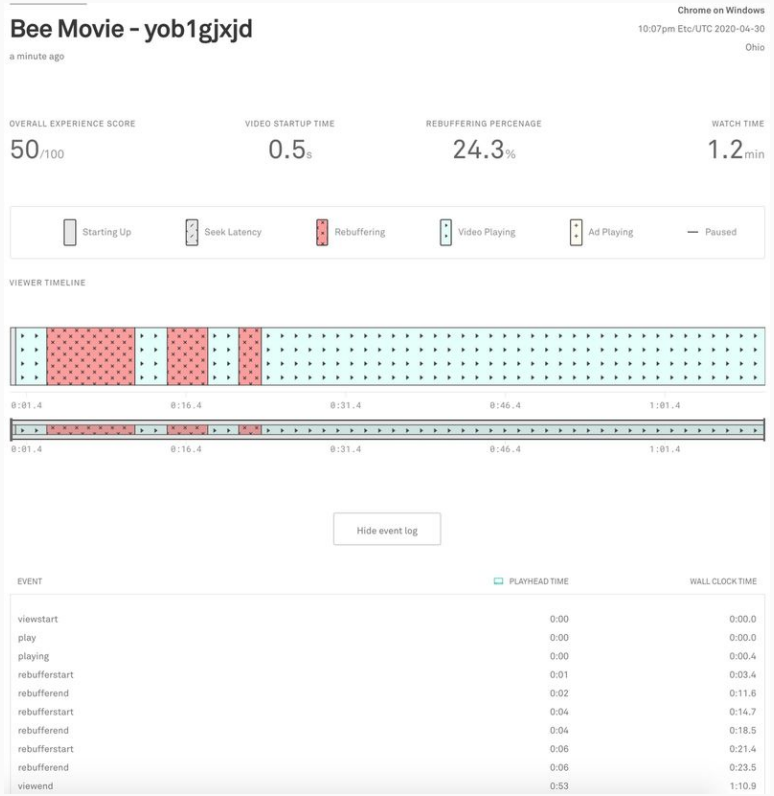
Altinity

Mux Data Overview



Video Views

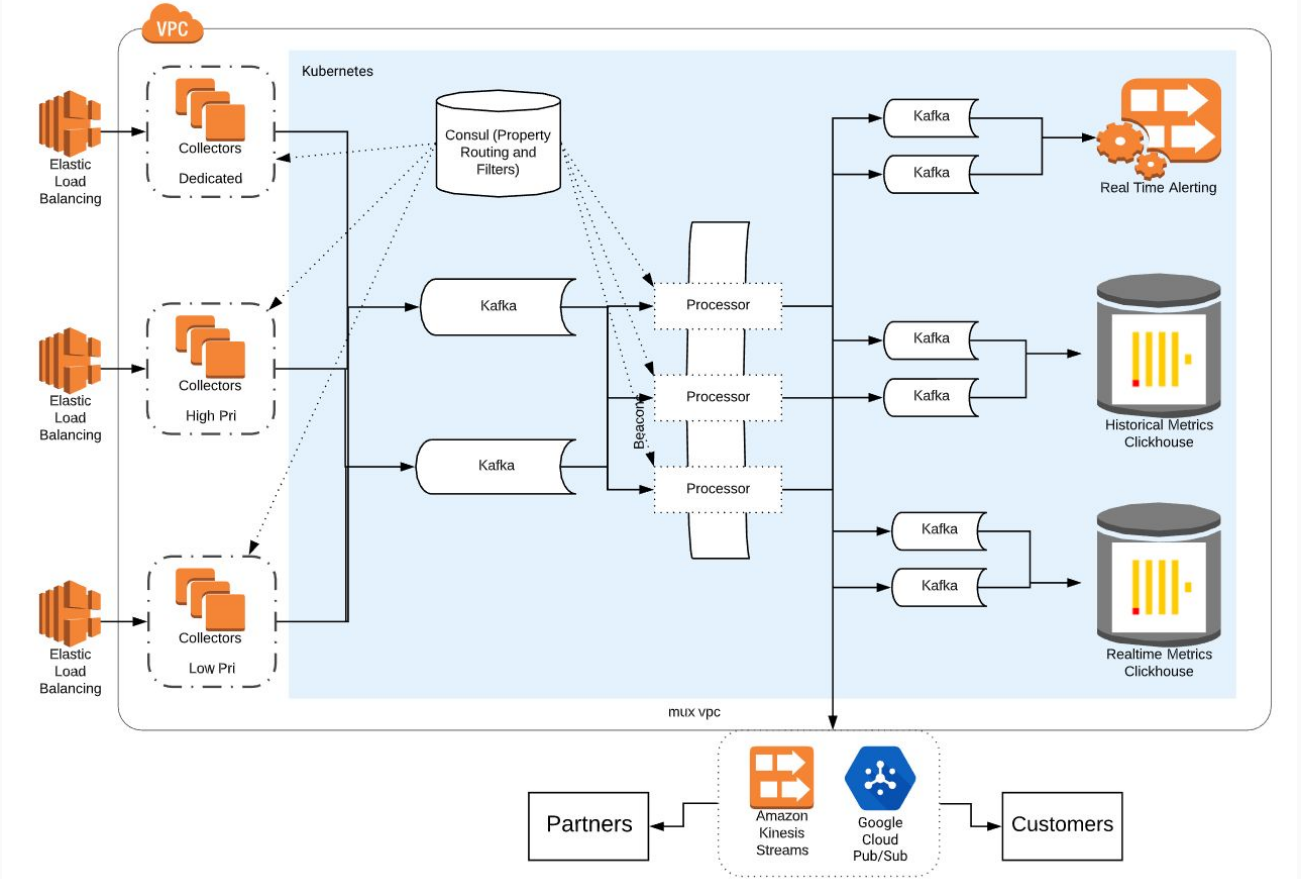




Altinity

MUX

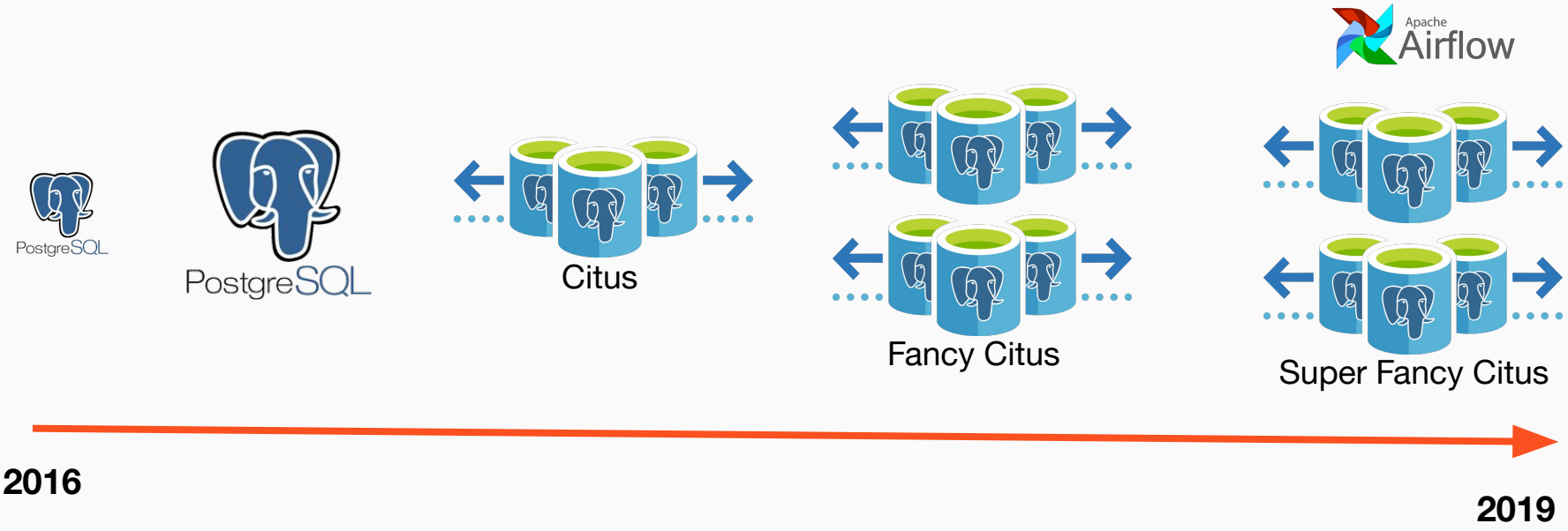
Data Architecture



Historical Metrics Clickhouse

- Full Video Views
- Billions of Views/Month
- 500M Views/Customer
- 100K Beacons/Second
- Raw Data Queries

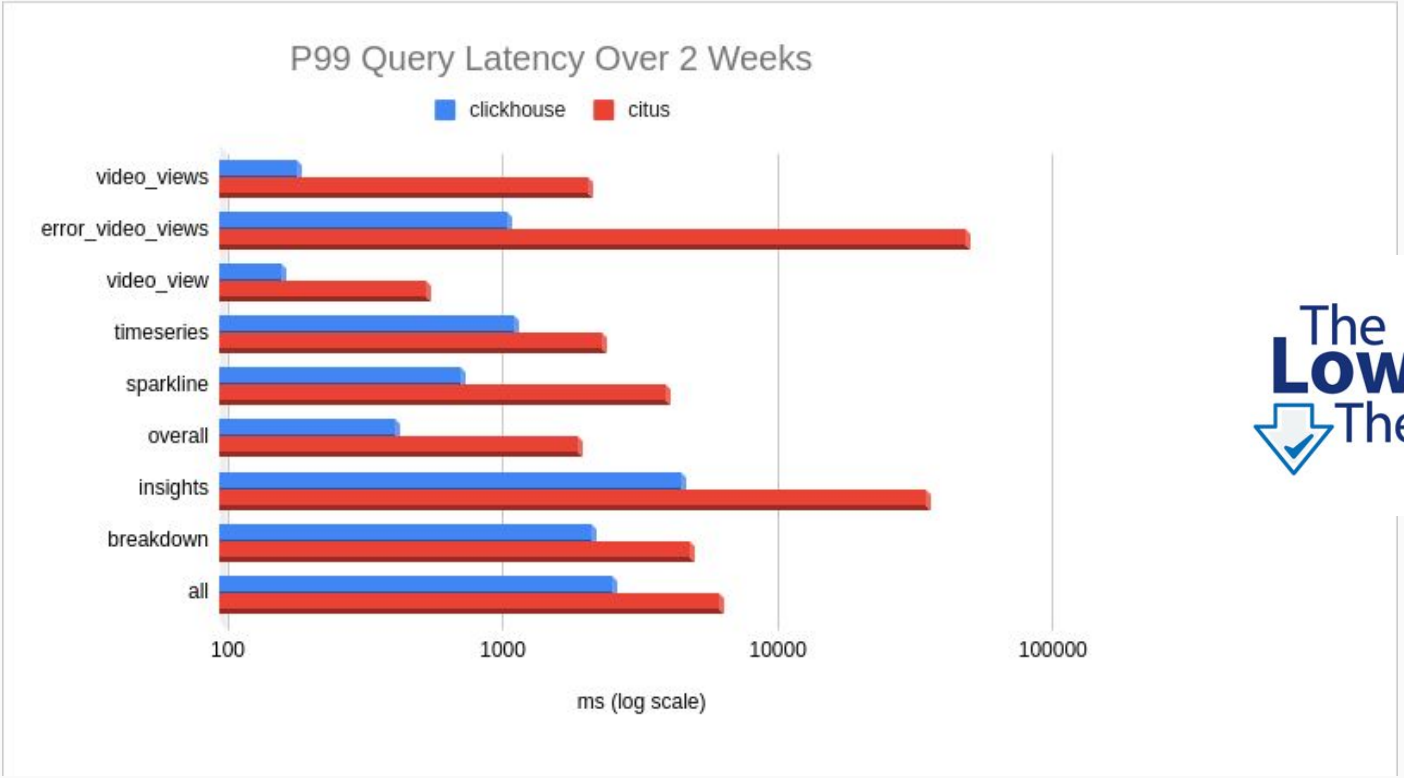
Mux Before Clickhouse



Unlocked Data Features

- Filter Depth
- Exclusion Filters
- Dynamic time aggregation
- Scale

Unlocked Performance



The Lower The Better

Cost Benefits

- No Aggregation (CPU + Disk)
- Columnar compression
  - Low Cardinality
  - ⅓ Disk Size
- Smaller machines
- Halved costs while volume doubled

Challenges

- Clickhouse != Postgres
- View Updates
- Individual record lookup

Updates

- “Resumed” views
- Low percentage of rows get updated
- **FINAL** keyword sometimes
  - Yes when listing views
  - No on larger metrics queries
  - Metrics workarounds
    - `SUM(metric * Sign) / SUM(Sign)`
- Nightly **OPTIMIZE**

```
CREATE TABLE views (
    view_id UUID,
    customer_id UUID,
    user_id UUID,
    view_time DateTime,
    ...
    rebuffer_count: UInt64,
    --- other metrics

    operating_system: Nullable(String),
    --- other filter dimensions

)
ReplicatedCollapsingMergeTree(...,Sign)
PARTITION BY toYYYYMMDD(view_time)
ORDER BY (customer_id, view_time, view_id)
```

Record Lookup

```
CREATE MATERIALIZED VIEW
     video_views_index_user_id
ON CLUSTER metrics
ENGINE = ReplicatedCollapsingMergeTree(...,
Sign)
PARTITION BY toYYYYMMDD(view_end)
PRIMARY KEY (property_id, user_id)
ORDER BY (property_id, user_id, view_end)
POPULATE AS
     SELECT
          property_id,
          user_id,
          view_end,
          Sign
     FROM video_views
```

```
SELECT * FROM video_views
WHERE
    view_time IN (
        SELECT view_time FROM user_id_index
        WHERE customer_id=0 AND user_id='mux'
    )
    AND customer_id=0 AND user_id='mux'
```

05/13/2020 (a month ago) 12 min read

From Russia With Love: How ClickHouse Saved our Data

by Kevin King | Engineering

The Mux Data platform is used by some of the biggest broadcasters to monitor the video streaming experience of their end users. Think of it like Google Analytics or New Relic for video playback. It's an essential tool that our customers rely on to make sure they are delivering smooth

https://mux.com/blog/from-russia-with-love-how-clickhouse-saved-our-data/

Nullable

Clickhouse Documentation

> **Note**
>
> Using `Nullable` almost always negatively affects performance, keep this in mind when designing your databases.

- Test it
- We use **Nullable** extensively
- Minimal Performance Impact

Deployment Details

- K8s
- 4 Clusters/4 Deployments
  - Historical Metrics
    - CH Replication
    - Secondary Cluster
    - 8 Node Clusters
  - Realtime Metrics
    - No Replication
    - Blue/Green Clusters
    - 5 Node Clusters
  - CDN logs
    - No Replication
    - Single Node
    - Increased Kafka retention
  - Raw Data Beacons
    - No Replication
    - 3 Day TTL

- All fronted by chproxy
  - https://github.com/Vertamedia/chproxy
  - Caching
  - User Routing
  - Rate Limiting
  - **Prometheus Monitoring**

What Next?

- Advanced Alerting
- BI Metrics
- Data Warehousing
- Move “beacon processing” into clickhouse

MUX

Wrap-up

Altinity

Takeaways

- Clickhouse performance feels like magic
- Operational simplicity, especially around scaling
- Clickhouse has become our default for statistic data

MUX

Thank you!

We are both hiring!

Mux:
https://mux.com

Altinity:
https://www.altinity.com

Altinity