Wolfram Wingerath

Big Data Analytics
With AWS Athena

Big Data
Big Data Analytics With AWS Athena

Wolfram Wingerath, Code.Talks 2019
I Am **Wolle**

**Research:**
- Stream Processing
- Real-Time Databases
- NoSQL & Cloud Systems
- ...

**Practice:**
- Web Caching
- Big Data Analytics
- Anger Management
- ...

Universität Hamburg

Baqend
I Like Real-Time Stuff

2018
Real-Time Processing Explained
A Survey of Storm, Samza, Spark & Flink

2017
Real-Time Databases Explained
Why Meteor, RethinkDB, Parse & Firebase Don't Scale
Let's Talk About Batch Analytics

Collection
- Raw PI tracking & meta data
- Custom tracking

Ingestion
- Tracking (RUM)

Analytics
- Materialized views & aggregations
- Historical data

SQL Interface

Reporting
- Performance Dashboard
- QA Dashboard
- Real-Time Alerting
- Ad-hoc SQL Interface
- Custom Reporting
What’s in the Data?
3 Things Make Your Website Slow

1. Backend Processing
2. Network Delays
3. Client
We Make Websites Fast

Website + **Speed Kit** (Service Worker)

Kino 7
Friday, 16:00

Fast Requests

**Speed Kit**
Cloud

Real-Time Sync

Tracking & Ad Services

Origin
Server
Accelerating Personalized HTML

Browser ➔ Merge ➔ Fast & Anonymous ➔ Speed Kit Cloud ➔ Origin Server

Slow & Personalized
Split Testing for Web Performance

Speed Kit Users
- Speed Kit enabled

Tracking

Measurable uplift:
- Performance
- User engagement
- Business success

Normal Users
- Speed Kit disabled (no acceleration)

Kino 7
Friday, 11:00
Goal: Performance & Business Insights

- Time-to-First-Byte
- First (Contentful) Paint
- DOM Timer
- First Input Delay

- Session Length
- Time on Site
- First User Interaction
- Bounce Rate

- Cart Size
- Transactions
- Conversion Rate
- Revenue

- Page Views & Sessions
- Browser Distribution
- JavaScript Errors
- Caching Insights

Browser

Cloud Backend

Tracking Beacon

Timing API
Service Worker
Unhandled Errors
How to Monitor Performance?

- Logging requests is not enough:
  - User? Rendering? ...
  - Browser cache (invisible)
  - Origin requests (no logs)
  - CDN requests

- Solution: Tracking every PI (page impression)
When to Send Data Beacons?

1. click & click detection
2. navigation & processing
3. user input

Promt for unload
redirect
App cache
DNS
TCP
Request
Response
Processing
onLoad
redirectStart
redirectEnd
fetchStart
domainLookupStart
domainLookupEnd
connectStart
connectEnd
requestStart
responseStart
responseEnd
loadEventStart
domComplete
domContentLoaded
domInteractive
domLoading
unloadEnd
unloadStart
navigationStart
(loadEventEnd)
1. 1 for **static** info
   (URL, user agent, session ID, ...)

2. 1 for **timings**
   (TTFB, load time, FCP, ...)

3. 0–n for **custom** events
   (first input, add-to-cart, ...)

**Metric Beacon**

**CustomEvent Beacon**
Let's Analyze the Data
Tracking Data in MongoDB

- Debugging with OLTP queries
- Analyses with aggregation pipeline, e.g.:
  - Average session length
  - Uplift vs. Acceleration
    (example: Conversion rate vs. first paint uplift)
  - Different browser timings by device/browser/...
CDN Logs in S3

- Automation via Jenkins jobs
- Caching-related statistics, e.g.:
  - Request latency distribution (histogram)
  - Image optimization efficiency
1. **Partitioning** by hour, but **not** by customer  
   → Not scalable!

2. **Indexing** & database import:  
   a) Import into indexed table  
   b) Drop index → import → create index  
   c) Query table without index  
   } each takes forever
Problems II: Aggregation Pipeline

**Indexing**
Queries over non-indexed attributes were infeasible

**Runtime**
Even with indexes in place, queries could take 30+ min.

**Scalability**
Queries got slower with increasing amounts of data

**Complexity**
MongoDB aggregation pipelines become sophisticated quickly
III: Reporting

- Scheduling:
  1. On-demand
  2. Periodic job

- Problems:
  - Cumbersome to build & maintain
  - Awkward to extend
  - Unreliable

- Automated reports
- Nice diagrams
- New problem
**Problems IV: Joins in MongoDB**

- **Joins infeasible** for data-intensive queries!
  - No conversion analysis
  - No business uplift validation
  - **Not acceptable!**
Fixing My Life With Flex Tape Athena
The „A“ Stands for „AWSome“

- Desperate attempt:
  1. Dump MongoDB collection
  2. Upload to S3
  3. Query with Athena

- Typical analysis:
  - 1 equi-join
  - 3 mio. Pls
  - ~15+ min.
The „A“ Stands for „AWSome“

- Desperate attempt: New best practice:
  1. Dump MongoDB collection
  2. Upload to S3
  3. Query with Athena

- Typical analysis:
  - 1 equi-join
  - 3 mio. Pls
  - ~10 seconds
What's an Athena?

- Managed Presto:
  - Interactive analytics with SQL
  - Heterogeneous datastores
  - Petabyte-scale (Facebook)

- Pricing by scanned data volume:
  - Efficient storage formats!
  - Partitioning or clustering!
  - Careful query design!

Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips et al. *Presto: SQL on Everything*, ICDE 2019
Upgrading Our ETL Pipeline

- **Simplicity:**
  - Everything in one place
  - Easy to access (SQL)

- **Scalability & efficiency:**
  - Hundreds of gigabytes scanned in a query
  - Response time on the order of seconds
Processing Stages & Latency

Alerting

- Simple metrics
  - Counters
  - Extreme values
  - Specific errors

Processing Stages

Trend Analysis

- Complex aggregations
  - Conversion rate
  - Performance by day
  - Seasonal effects
Stage 0: Data Preparation

```json
{
    "_id": "ABC",
    "loadEvent": {
        "$numberLong": "1571101211368"
    },
    "createdAt": {
        "$date": "2019-10-15T01:00:11.462Z"
    },
    ...,
}
```

- Schema Definition
  1. **Tables** for raw data
  2. **Views** on top to hide artifacts

<table>
<thead>
<tr>
<th></th>
<th>_id</th>
<th>loadEvent</th>
<th>createdAt</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>Oct. 15, 2019&lt;br&gt;1h 0m 11s 368ms</td>
<td>Oct. 15, 2019&lt;br&gt;1h 0m 11s 462ms</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example: Timestamps

```sql
FROM_UNIXTIME(
    CAST(
        CAST(
            JSON_EXTRACT(loadEventEndRaw, '$"$numberlong"')
        AS VARCHAR
    AS DECIMAL (38,3)) / 1000
) AS loadEventEnd
```

1. Extract UNIX timestamp from JSON
2. Cast to varchar
3. Cast to decimal
4. Divide by 1000
5. Convert to timestamp
CASE WHEN
    CAST(
        CAST(
            JSON_EXTRACT(loadEventEndRaw, '$"numberlong"')
        AS VARCHAR)
    AS DECIMAL (38,3)) > 0
THEN
    FROM_UNIXTIME(
        CAST(
            CAST(
                JSON_EXTRACT(loadEventEndRaw, '$"numberlong"')
            AS VARCHAR)
        AS DECIMAL (38,3)) / 1000
    )
END AS loadEventEnd

Note: still a timestamp (no timing in ms)
### Stage 1: **Join Beacons**

<table>
<thead>
<tr>
<th>Metric</th>
<th>CustomEvent</th>
<th>Metric_complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>User/Session (URL, user ID, ...)</td>
<td>• Behavior &amp; UX (conversion, first input, JS errors, ...)</td>
<td>• User/session</td>
</tr>
<tr>
<td>Timers (TTFB, FP, ...)</td>
<td></td>
<td>• Timers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Behavior</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UX</td>
</tr>
</tbody>
</table>

- Consolidate PI data into **single rows**
- Data **cleaning**
  - (e.g. nullify when `loadEnd < navigationStart`)
Stage 2: Resolve User Agents

- **Metric_complete**
  - User agent
  - ...

- **Metric_complete**
  - browser (+ version)
  - Device type
  - Device
  - OS (+ version)
  - ...

WhatIsMyBrowser.com

- **Paid service**: interpreting user agents is complex!
- **Fallback**: simple `case-when` logic for browsers and device type
- **Simplification** required, e.g.:
  - Device: mobile/desktop/tablet/server/game console/wearable/vehicle/...
  - Browser: Chrome/Firefox/Safari/Opera/IE/Netscape/Tesla-Browser/...
Stage 3: **Session Metrics**

- **Aggregation** by session ID
  - Session length, bounces
  - Time on site
  - Performance (e.g. median FCP)
  - Conversions
Stage 4: **Materialized Views**

- Business uplift during Proof of Concept (PoC)
- Aggregation over days or weeks
- Performance and business trend analysis
Reporting: The Right Tool for the Job

Custom Code

Automated reports  Nice diagrams  New problem

Unified analytics
Reporting: The Right Tool for the Job

- Requirements:
  - Automation
  - Easy data exploration
  - Robustness

Custom Code

Athena

Simple Reporting

- Automated reports
- Nice diagrams
- New problem
Why QuickSight?

- Easy integration with Athena
- Ease-of-use
- Quick results

AWS Services:
- S3
- Athena
- QuickSight

Diagram showing integration with Fastly and Baqend.
Why, Oh, Why, QuickSight?

Speed Kit PoC Analysis Dashboard

**Executive Summary of Uplifts (Positive Value = Improvement)**

<table>
<thead>
<tr>
<th>skgroup</th>
<th>Last X days (Including Yesterday)</th>
<th>Device Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKHTML</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>last 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sessions</td>
<td>all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mobile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Desktop</td>
</tr>
<tr>
<td></td>
<td>Orders</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Conversations</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Order Value</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shed Order</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Order Uplift</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shed Order Uplift</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Order Value %</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Load Uplift</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>First Paint Up</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Session Length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Return Rate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bounce Impor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADV Uplift</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td># FV uplift</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#Session Uplift</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#User Uplift</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>addToCart Vol</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#addToCart</td>
<td></td>
</tr>
</tbody>
</table>
### Speed Kit PoC Analysis Dashboard

<table>
<thead>
<tr>
<th>Skyscraper</th>
<th>Last X Days (including yesterday)</th>
<th>Device Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>all</td>
</tr>
<tr>
<td>Sessions</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Orders</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Conversions</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Order Value</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>viewed Order</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Order Uplift</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>viewed Order</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Order Value u</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Load Time</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>First Paint Up</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Session Length</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Return Rate</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Bounces Imp</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>ADV Uplift</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td># Pit Uplift</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td># Session 01</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td># User Uplift</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>addToCart Val</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td># addToCart</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Why, Oh, Why, QuickSight?**
Time Works Differently in AWS Dashboards

Running Query...

Estimated time elapsed: 0 seconds

You can run another query by clicking on the New Query button. The current query will continue to run in the background. You can check the status of all queries in the History Tab.
Time Works Differently in AWS Dashboards

Congratulations! Your query took 2 seconds only!

You can run another query by clicking on the New Query button. The current query will continue to run in the background. You can check the status of all queries in the History Tab.
Click Once, Pay Twice

1. You haven’t updated the data set, yet.

2. Something went wrong.
3. Unable to retrieve preview data from the source within 30 seconds. Please try again or prepare and visualize the data without preview.
4. Reload preview.

/* QuickSight c26063a5-b35a-4682-a6d8-e5e7f0a1e1f */ SELECT "app", "days_in_past", "responsecause",...
But There is More!

- **Questionable Limits**
  Diagrams in the dashboard, query timeouts, etc.

- **Exhausted Resources** in Athena
  Queries fail depending on time, day, and weather

- **Implicit Query Rewriting** in QuickSight
  Hard to predict, hard to circumvent, hardly helpful
Real-Time Analytics & Reporting

- Next Steps:
  - Continuous ingestion
  - Streaming analytics & real-time dashboards
  - Drop QuickSight
Real-Time Analytics & Reporting

- Next Steps:
  - Continuous ingestion
  - Streaming analytics & real-time insights
  - Drop QuickSight
Thanks! Any Questions?

Kino 3
Thursday, 16:00  
GDPR Panel Discussion

Kino 7
Friday, 11:00  
Web Performance Talk

Kino 7
Friday, 16:00  
Service Worker Hackathon
Thanks! Any Questions?

Right Here Right Now

- GDPR Panel Discussion

Kino 7
Friday, 11:00

- Web Performance Talk

Kino 7
Friday, 16:00

- Service Worker Hackathon

Wolfram Wingerath, Data Engineering

ww@baqend.com

Baqend