# Swimming in the Data River

**Jon King**
jon.king@imply.io

# Who am I?

Jon King

13 years in Big Data (Data Lakes, Hadoop, Hive, Druid)

Principal Field Architect @ 


O'reilly Contributor and Author

- Operationalizing the data lake in the cloud (2019) - Ackerman & King
- Programming Hive (2012) - Capriolo, Wampler & Rutherglen

# Agenda

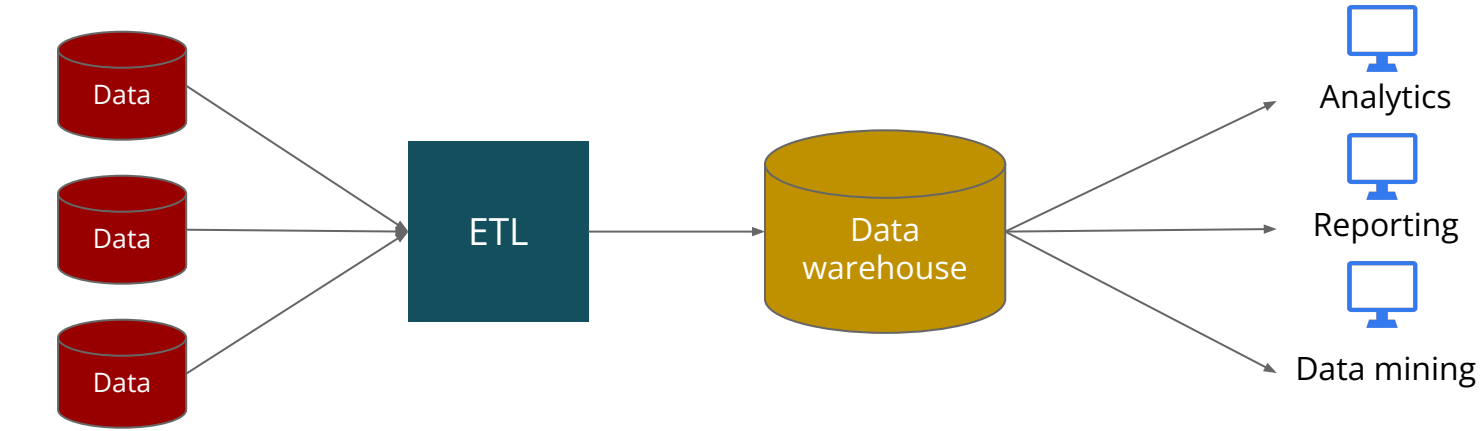
- From warehouses to rivers
- Analytics problems at scale
- Enter the Druid
- Druid deep dive
- Do try this at home!

# Rolling down the river

# Let’s go back to where it all began

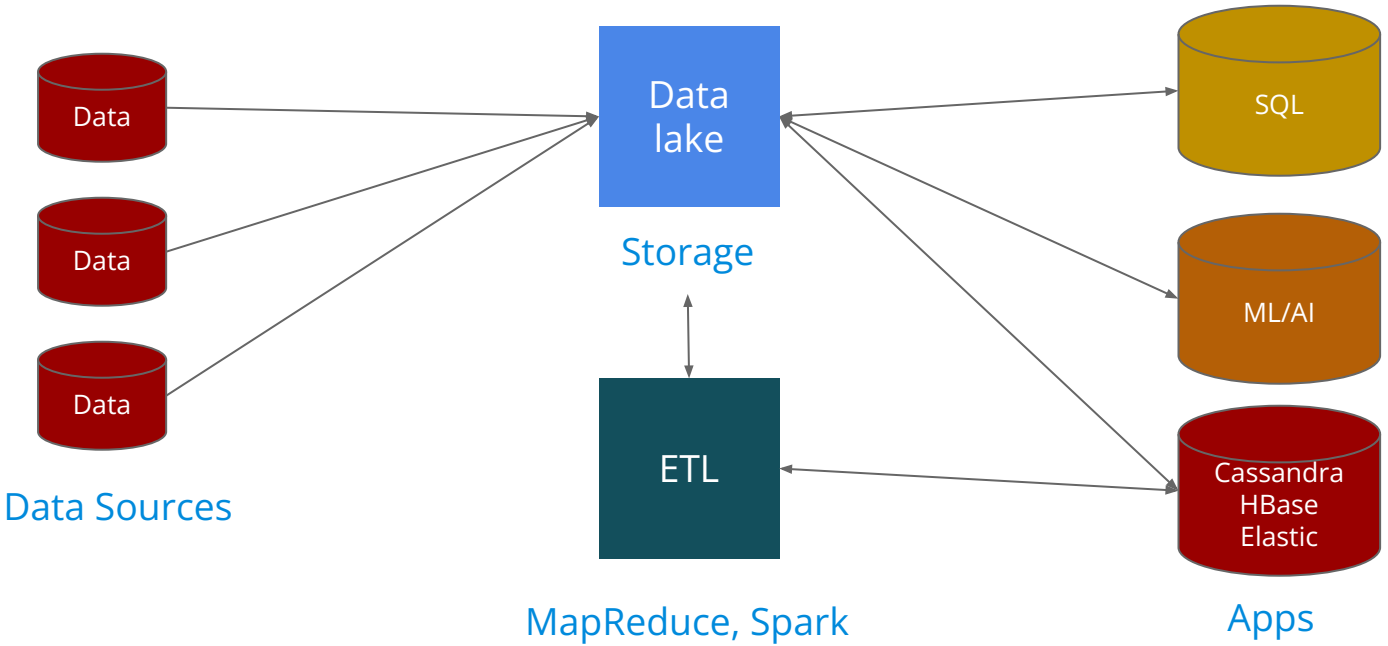
# Data warehouses

Tightly coupled architecture with limited flexibility.



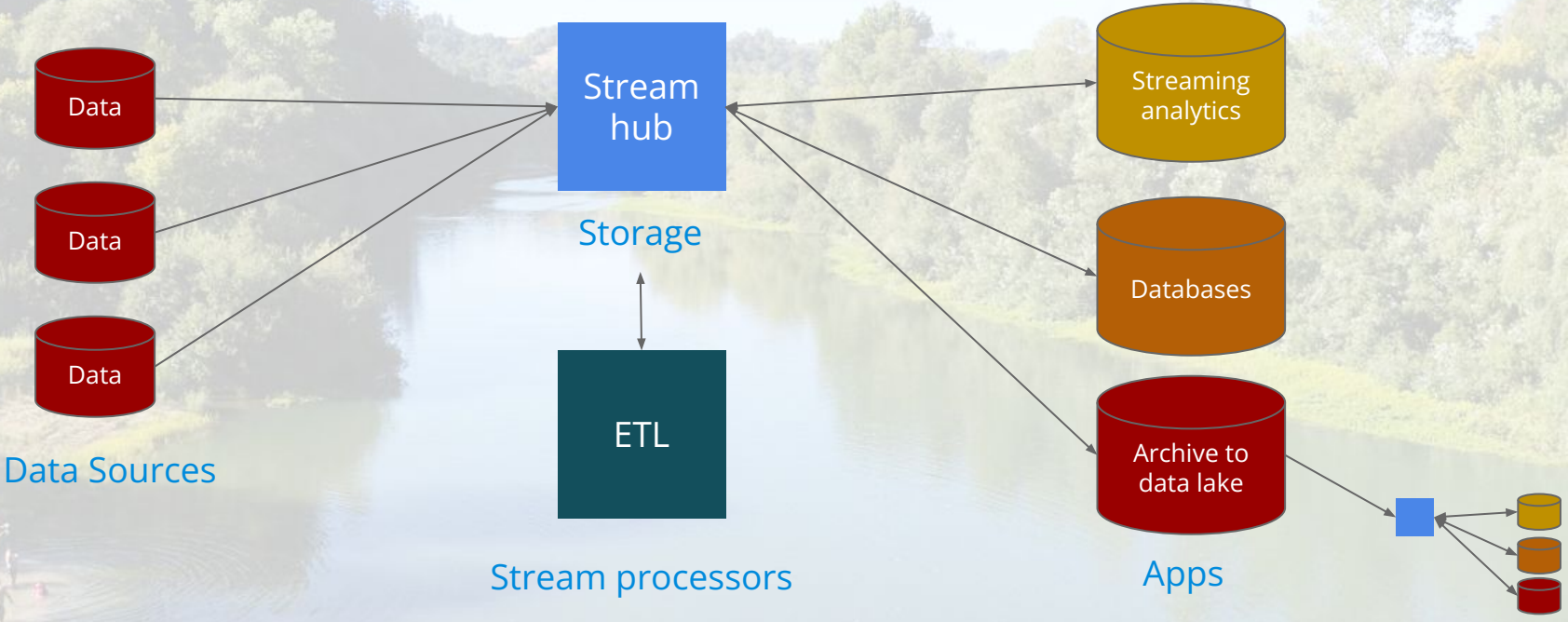
# Data lakes

Modern data architectures are more application-centric.



# Data rivers

Streaming architectures are true-to-life and enable faster decision cycles.



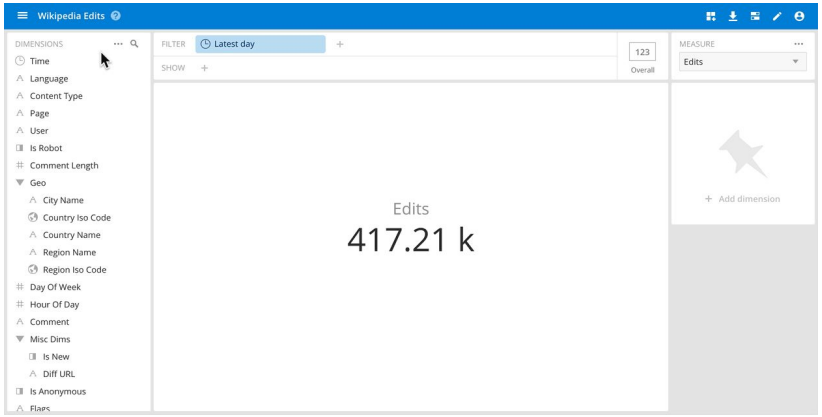
# The problem

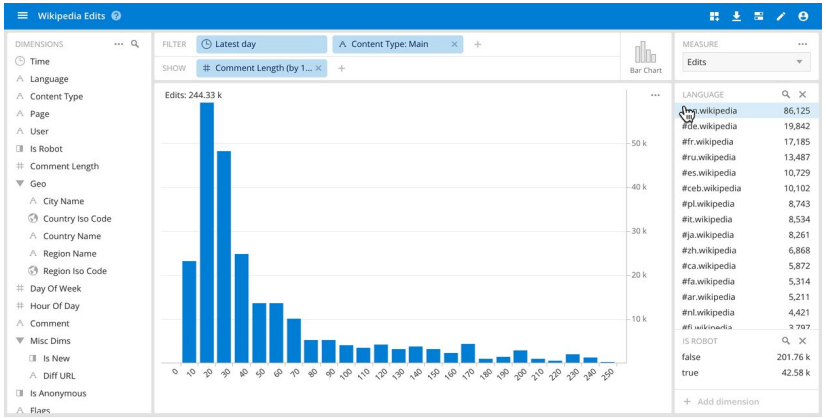

DevOps Borat @DEVOPS_BORAT · 23 Mar 2013
In startup we have great of capability for churn out solution. Please send problem, we are pay good money.
71
28

# The problem

- Slice-and-dice for big data streams
- Interactive exploration
- Look under the hood of reports and dashboards
- And we want our data fresh, too

# What the end users want

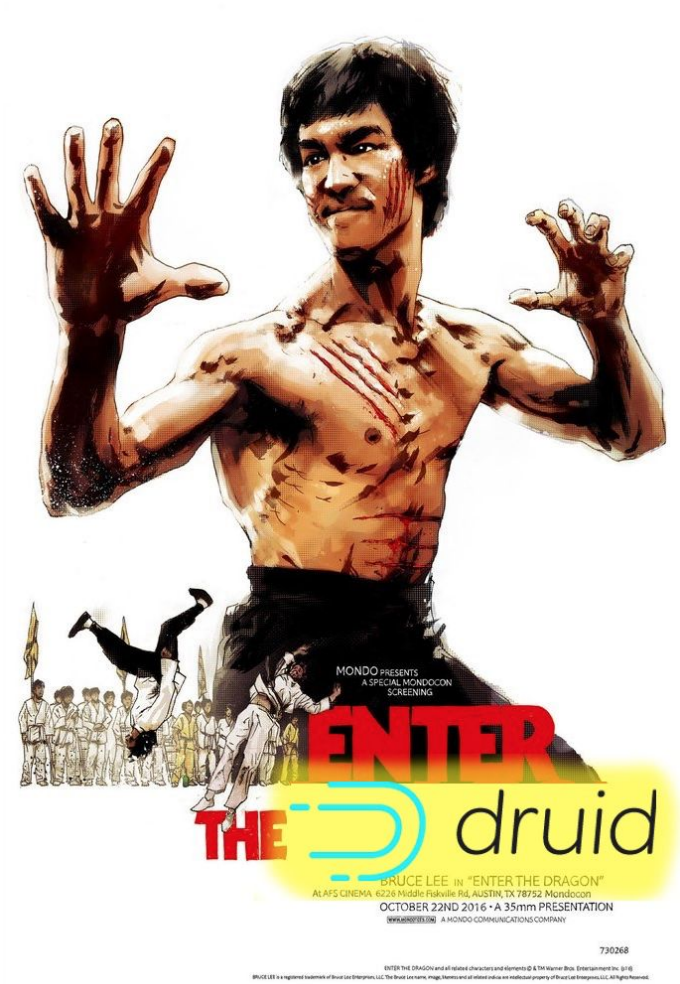



# Challenges

- Scale: when data is large, we need a lot of servers
- Speed: aiming for sub-second response time
- Complexity: too much fine grain to precompute
- High dimensionality: 10s, 100s or 1000s of dimensions
- Concurrency: many users and tenants
- Freshness: load from streams

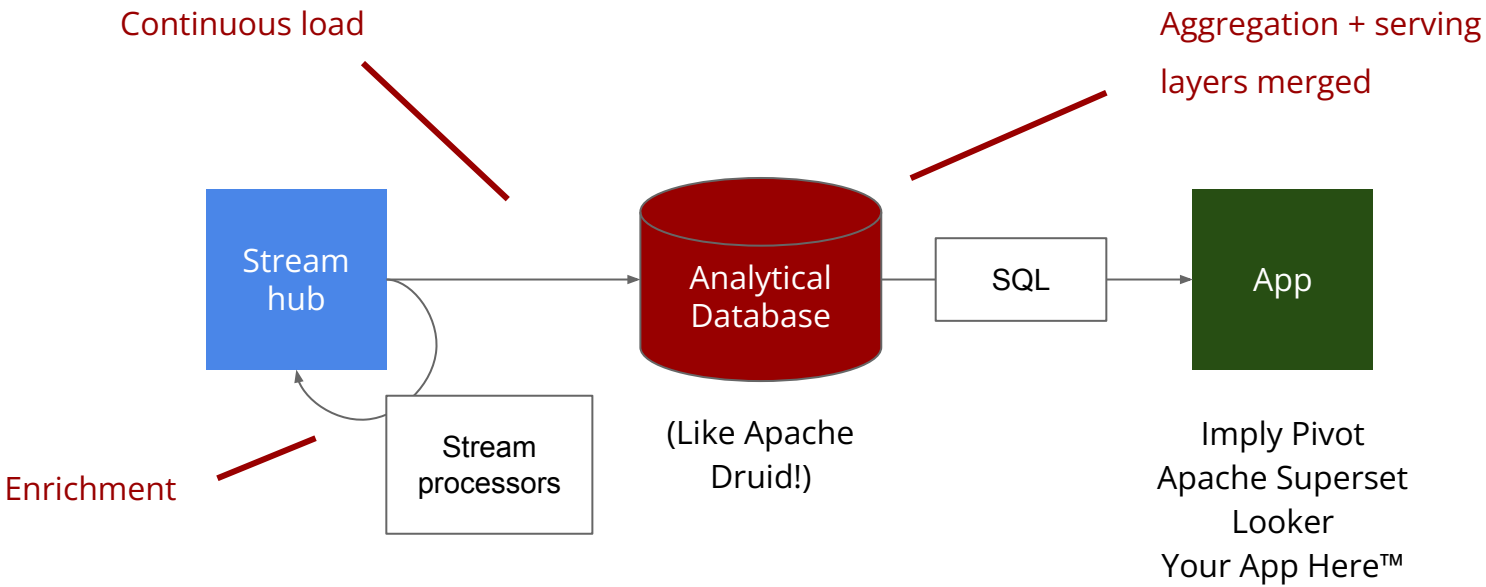
high performance analytics data store for event-driven data



# What is Druid?

- **“high performance”:** low query latency, high ingest rates
- **“analytics”:** counting, ranking, groupBy, time trend
- **“data store”:** the cluster stores a copy of your data
- **“event-driven data”:** fact data like clickstream, network flows, user behavior, digital marketing, server metrics, IoT

# Streaming data



# Key features

- Column oriented
- High concurrency
- Scalable to 100s of servers, millions of messages/sec
- Continuous, real-time ingest
- Indexes on all dimensions by default
- Query through SQL
- Target query latency sub-second to a few seconds

# Use cases

- Clickstreams, user behavior
- Digital advertising
- Application performance management
- Network flows
- IoT

# Powered by Apache Druid


*Source: http://druid.io/druid-powered.html*

# Powered by Apache Druid


*Source: http://druid.io/druid-powered.html*

# Powered by Apache Druid

From Yahoo:

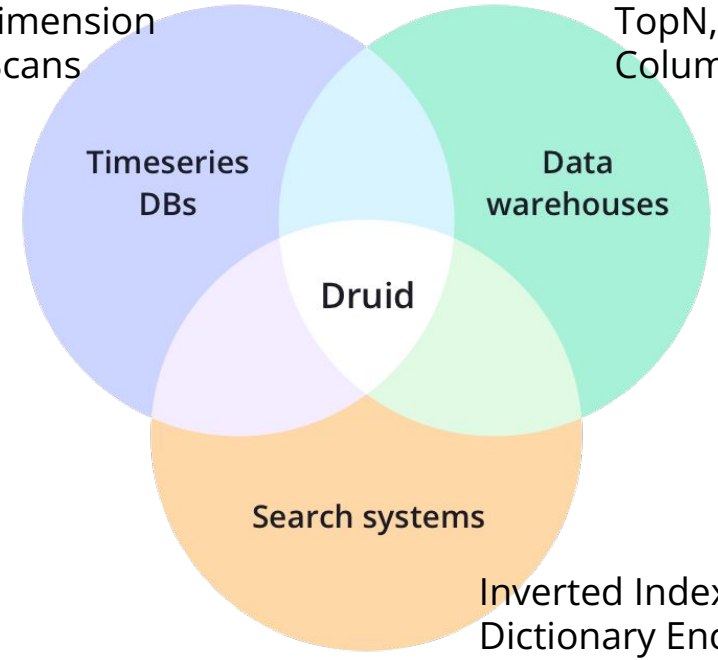
"The performance is great … some of the tables that we have internally in Druid have **billions and billions of events** in them, and we're scanning them in **under a second**."

*Source: https://www.infoworld.com/article/2949168/hadoop/yahoo-struts-its-hadoop-stuff.html*

# Best of Breed Architectures

Group By Time
Partition by dimension
Time bound Scans

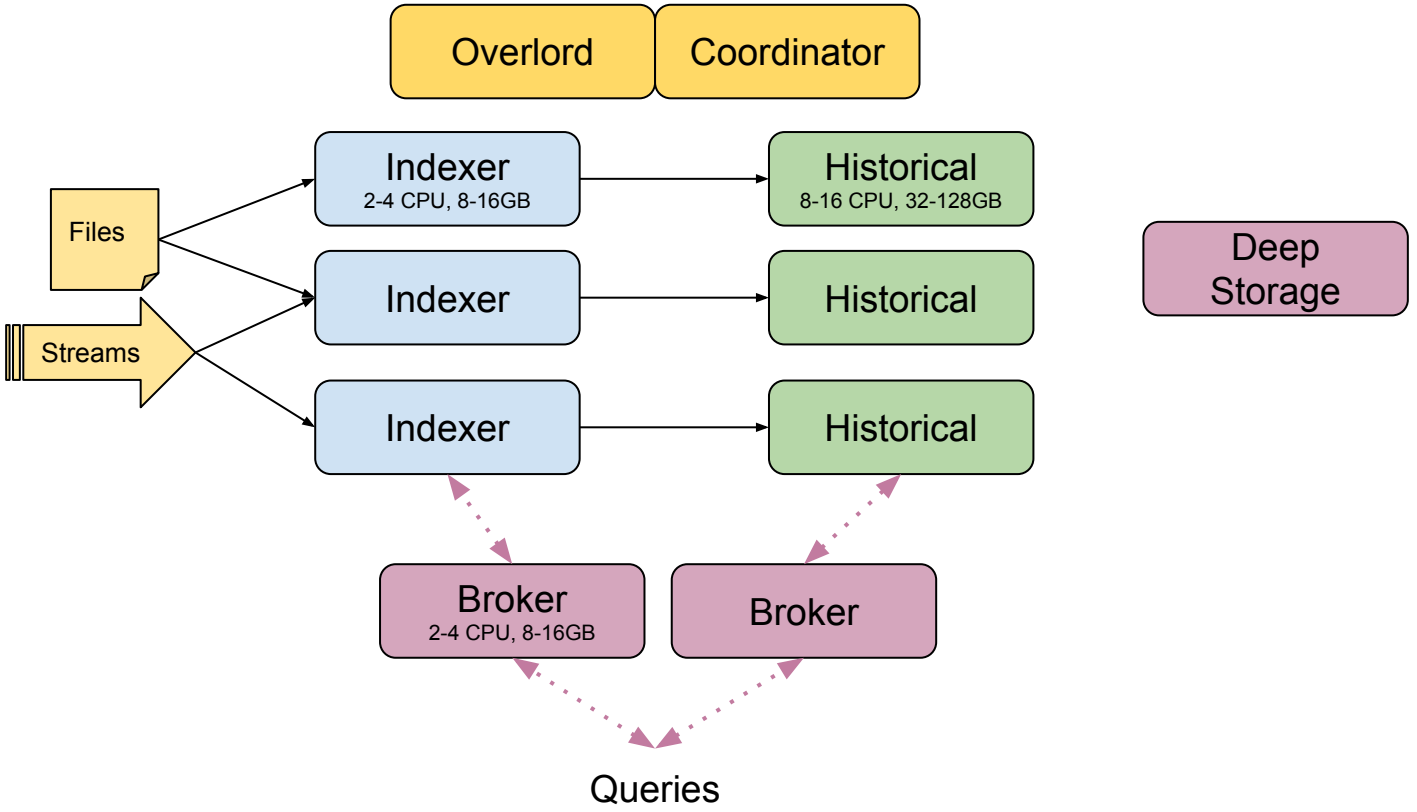
OLAP Analytics
TopN, GroupBy, Timerseries
Column oriented




Inverted Indexes
Dictionary Encoding
Bitmap Indexes

# How It works

# Microservice Architecture



# Druid Processes

| | |
|---|---|
| Coordinator Nodes | Segment management and distribution |
| Broker Nodes | Route queries and merge results |
| Historical Nodes | Loading and serving of segments |
| Overlord Nodes | Responsible for watching over and delegating index tasks |
| Middle Manager / Indexer | Data ingestion (indexing), serving real-time queries for broker |

# Where does Druid fit best?

- Arbitrary slicing and dicing of very large datasets
- Real-time analysis
- Sub-Second Fast interactive response e.g. BI, Analytics
  - Behavior analytics e.g. unique visitors per hour, per day, retention analysis
  - Experiments e.g. root cause analysis
  - Web traffic analysis by time

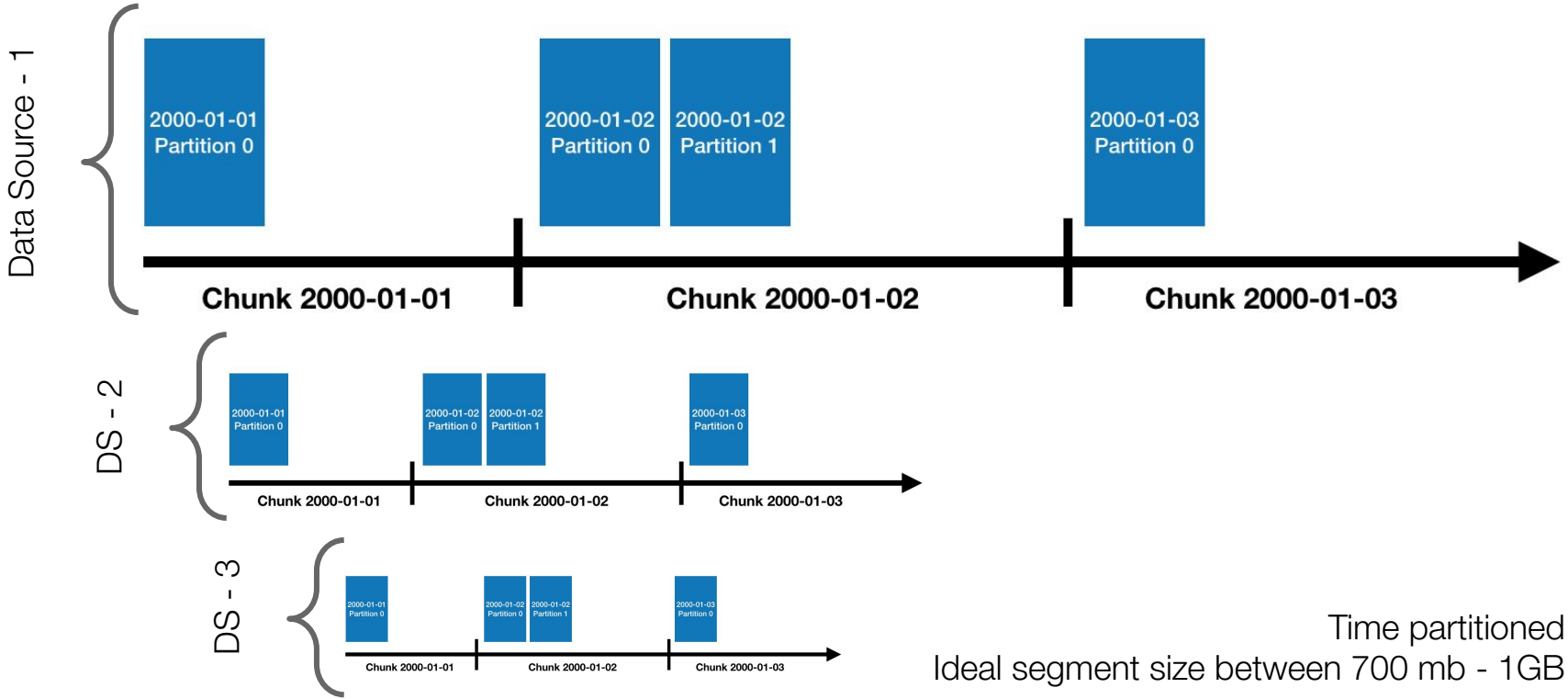
Funnel Analysis

Network Traffic Analysis

User Behavior Analysis

A/B Test Analysis

# Datasources Deep Storage and Segments




Time partitioned ⇐

Ideal segment size between 700 mb - 1GB ⇐

# Encoding and Indexing Process

| timestamp | page | language | city | country | added | deleted |
|---|---|---|---|---|---|---|
| 2011-01-01T00:01:35Z | Ben Hopp | en | SF | USA | 10 | 22 |
| 2011-01-01T00:02:14Z | Ben Hopp | en | SF | USA | 2 | 11 |
| 2011-01-01T00:03:38Z | Ben Hopp | en | SF | USA | 6 | 10 |
| 2011-01-01T00:05:45Z | Andre | en | LA | CA | 15 | 23 |
| 2011-01-01T00:06:15Z | Andre | en | LA | CA | 12 | 17 |
| 2011-01-01T00:07:05Z | Harry Gomes | en | LA | CA | 18 | 65 |


| page | id |
|---|---|
| Ben Hopp | 0 |
| Andre | 1 |
| Harry Gomes | 2 |


| | |
|---|---|
| page | [ 0 0 0 1 1 2 ] |

- Each value is stored as bitmap index
- Indexes are compressed

# Encoding and Indexing Process

## Bitmap Indices

| timestamp | page | language | city | country | ... | added | deleted |
|---|---|---|---|---|---|---|---|
| 2011-01-01T00:01:35Z | Justin Bieber | en | SF | USA | | 10 | 65 |
| 2011-01-01T00:03:63Z | Justin Bieber | en | SF | USA | | 15 | 62 |
| 2011-01-01T00:04:51Z | Justin Bieber | en | SF | USA | | 32 | 45 |
| 2011-01-01T00:01:35Z | Ke$ha | en | Calgary | CA | | 17 | 87 |
| 2011-01-01T00:01:35Z | Ke$ha | en | Calgary | CA | | 43 | 99 |
| 2011-01-01T00:01:35Z | Selena Gomes | en | Calgary | CA | | 12 | 53 |

- Store Bitmap Indices for each value
  - Justin Bieber -> [0, 1, 2] -> [1 1 1 0 0 0]
  - Ke$ha -> [3, 4] -> [0 0 0 1 1 0]
  - Selena Gomes -> [5] -> [0 0 0 0 0 1]
- Queries
  - Justin Bieber or Ke$ha -> [1 1 1 0 0 0] OR [0 0 0 1 1 0] -> [1 1 1 1 1 0]
  - language = en and country = CA -> [1 1 1 1 1 1] AND [0 0 0 1 1 1] -> [0 0 0 1 1 1]
- Indexes compressed with Concise or Roaring encoding

# Auto Summarization - “The Rollup Thing”

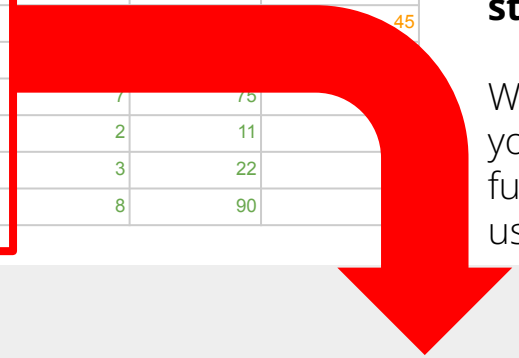
| timestamp | page | language | city | country | added | deleted |
|---|---|---|---|---|---|---|
| 2011-01-01T00:01:35Z | Ben Hopp | en | SF | USA | 10 | 22 |
| 2011-01-01T00:02:14Z | Ben Hopp | en | SF | USA | 2 | 11 |
| 2011-01-01T00:03:38Z | Ben Hopp | en | SF | USA | 6 | 10 |
| 2011-01-01T00:05:45Z | Andre | en | LA | CA | 15 | 23 |
| 2011-01-01T00:06:15Z | Andre | en | LA | CA | 12 | 17 |
| 2011-01-01T00:07:05Z | Harry Gomes | en | LA | CA | 18 | 65 |

| timestamp | page | language | city | country | count | sum_added | sum_deleted | min_added | max_added |
|---|---|---|---|---|---|---|---|---|---|
| 2011-01-01T00:00:00Z | Ben Hopp | en | SF | USA | 3 | 18 | 43 | 2 | 10 |
| 2011-01-01T00:00:00Z | Andre | en | LA | CA | 2 | 27 | 40 | 12 | 15 |
| 2011-01-01T00:00:00Z | Harry Gomes | en | LA | CA | 1 | 18 | 65 | 18 | 18 |

# Approximations?

# Sketches + rollup = BFF

| timestamp | page | city | userid | count | sum_added | sum_deleted |
|---|---|---|---|---|---|---|
| 2011-01-01T00:01:35Z | Justin Bieber | SF | user11 | 8 | 10 | 5 |
| 2011-01-01T00:03:45Z | Justin Bieber | SF | user22 | 3 | 25 | 37 |
| 2011-01-01T00:05:62Z | Justin Bieber | SF | user11 | 4 | 15 | 19 |
| 2011-01-01T00:06:33Z | Ke$ha | LA | user33 | | | 45 |
| 2011-01-01T00:08:51Z | Ke$ha | LA | user33 | | | |
| 2011-01-01T00:09:17Z | Miley Cyrus | DC | user11 | 7 | 75 | |
| 2011-01-01T00:11:25Z | Miley Cyrus | DC | user44 | 2 | 11 | |
| 2011-01-01T00:23:30Z | Miley Cyrus | DC | user44 | 3 | 22 | |
| 2011-01-01T00:49:33Z | Miley Cyrus | DC | user55 | 8 | 90 | |



| timestamp | page | city | | | | userid_sketch | count | sum_added | sum_deleted |
|---|---|---|---|---|---|---|---|---|---|
| 2011-01-01T00:00:00Z | Justin Bieber | SF | | | | {sketch_data structure} | 15 | 50 | 61 |
| 2011-01-01T00:00:00Z | Ke$ha | LA | | | | {sketch_data structure} | 10 | 46 | 53 |
| 2011-01-01T00:00:00Z | Miley Cyrus | DC | | | | {sketch_data structure} | 20 | 198 | 88 |

Sketches can also be generated ***during ingestion***. This isn't required, but if you do it, there are strong potential synergies between sketches and rollup.

If sketching allows you to **skip storing the original column**, then it can **improve rollup ratio** and **avoid storing PII**.

When querying a sketched column, you must use a sketch aggregation function that matches the one you used during ingestion.

# Sketches

**HyperLogLog**

Theta

Quantiles

Sketch Errors

# HyperLogLog is great for…

## Distinct counts

### About HyperLogLog

Incredibly popular sketch for DISTINCT COUNTs

Alternative to *COUNT(DISTINCT expr)*

Widely used in the big data space

Fast, memory-efficient, quite accurate

### Usage notes

- Druid has two HLL implementations: builtin and DataSketches.
  - Builtin: *COUNT(DISTINCT [expr])* and *APPROX_COUNT_DISTINCT* in SQL; "hyperUnique" and "cardinality" in JSON
  - DataSketches: *APPROX_COUNT_DISTINCT_DS_HLL* in SQL; "HllSketchBuild" and "HllSketchMerge" in JSON
- Use the one from DataSketches: it is **faster** and **more accurate**.
- Use *APPROX_COUNT_DISTINCT_DS_HLL(expr)* instead of *COUNT(DISTINCT [expr])*.

# HyperLogLog is great for…

## Distinct counts

| lgK | # Std Dev ➡ / Confidence interval ➡ / bytes | 1 / 68.27% ⬇ | 2 / 95.45% ⬇ | 3 / 99.73% ⬇ | |
|---|---|---|---|---|---|
| 10 | 616 | 3.25% | 6.50% | 9.75% | |
| 11 | 1,128 | 2.30% | 4.60% | 6.89% | |
| **12** | **2,216** | **1.63%** | **3.25%** | **4.88%** | ⬅ *default* |
| 13 | 4,264 | 1.15% | 2.30% | 3.45% | |
| 14 | 8,488 | 0.81% | 1.63% | 2.44% | |
| 15 | 16,936 | 0.57% | 1.15% | 1.72% | |
| 16 | 33,832 | 0.41% | 0.81% | 1.22% | |
| 17 | 67,624 | 0.29% | 0.57% | 0.86% | |
| 18 | 135,208 | 0.20% | 0.41% | 0.61% | |
| 19 | 270,376 | 0.14% | 0.29% | 0.43% | |
| 20 | 540,712 | 0.10% | 0.20% | 0.30% | |
| 21 | 1,081,384 | 0.07% | 0.14% | 0.22% | |

**Size / error table**

# Sketches

HyperLogLog

**Theta**

Quantiles

Sketch Errors

# Theta is great for…

## Distinct counts with *intersection* and *difference*

### About Theta sketches

Count the overlap between sets

Useful for retention analysis

Useful for *order-independent* funnel analysis

### Usage notes

- vs. HLL: Theta uses *30x more memory* and (as of 2021.05) Theta is *slightly slower than HLL*.
  Prefer HLL when you don't need intersection and difference.
- Error of small sets derived from large sets is higher than specified in the table.
  Example: *THETA_SKETCH_INTERSECT(theta1, theta2)* when theta1 and theta2 have very little overlap.
  Example: *THETA_SKETCH_INTERSECT(theta1, theta2)* when theta1 is large and theta2 is small.
  In general, error anchors onto the size of the *union* of *all sets* involved in a series of set ops.

# Theta is great for…

## Distinct counts with *intersection* and *difference*

| | # Std Dev ➡ | 1 | 2 | 3 | | Size / error table |
|---|---|---|---|---|---|---|
| | Confidence interval ➡ | 68.27% | 95.45% | 99.73% | | |
| size | bytes | ⬇️ | ⬇️ | ⬇️ | | |
| 1,024 | 16,384 | 3.25% | 6.50% | 9.75% | | |
| 2,048 | 32,768 | 2.30% | 4.60% | 6.89% | | |
| 4,096 | 65,536 | 1.63% | 3.25% | 4.88% | | |
| 8,192 | 131,072 | 1.15% | 2.30% | 3.45% | | |
| **16,384** | **262,144** | **0.81%** | **1.63%** | **2.44%** | ⬅️ *default* | |
| 32,768 | 524,288 | 0.57% | 1.15% | 1.72% | | |
| 65,536 | 1,048,576 | 0.41% | 0.81% | 1.22% | | |
| 131,072 | 2,097,152 | 0.29% | 0.57% | 0.86% | | |
| 262,144 | 4,194,304 | 0.20% | 0.41% | 0.61% | | |
| 524,288 | 8,388,608 | 0.14% | 0.29% | 0.43% | | |
| 1,048,576 | 16,777,216 | 0.10% | 0.20% | 0.30% | | |
| 2,097,152 | 33,554,432 | 0.07% | 0.14% | 0.22% | | |

# Sketches

HyperLogLog

Theta

**Quantiles**

Sketch Errors

# Quantiles are great for...

## Approximate medians, percentiles, ranks, and histograms

### About Quantiles sketches

Useful for analyzing the distribution of numeric values.

### Usage notes

- Druid has four (!!) implementations: Approximate Histogram, DataSketches, t-digest, and Moment.
  - Approximate Histogram: *APPROX_QUANTILE* in SQL; "approxHistogram" and "approxHistogramFold" in JSON
  - DataSketches: *APPROX_QUANTILE_DS* in SQL; "quantilesDoublesSketch" in JSON
  - t-digest: *TDIGEST_QUANTILE* in SQL; "tDigestSketch" in JSON
  - Moment: "momentSketch" and "momentSketchMerge" in JSON
- Use the one from DataSketches: it is **fast** and **accurate** and is a fully-supported **core extension**.
- Approximate Histogram is also a core extension, but it uses an older, less accurate algorithm.
- t-digest and Moment are contrib extensions and are not supported by Imply.

# Quantiles are great for…

## Approximate medians, percentiles, ranks, and histograms

| K | bytes at start | bytes after 4B items | ~ Error | |
|---|---|---|---|---|
| 16 | 3,488 | 3,744 | 12.145% | |
| 32 | 6,688 | 7,200 | 6.359% | |
| 64 | 12,832 | 13,856 | 3.317% | |
| **128** | **24,608** | **26,656** | **1.725%** | ⬅ *default* |
| 256 | 47,136 | 51,232 | 0.894% | |
| 512 | 90,144 | 98,336 | 0.463% | |
| 1,024 | 172,064 | 188,488 | 0.239% | |
| 2,048 | 327,712 | 360,480 | 0.123% | |
| 4,096 | 622,624 | 688,160 | 0.063% | |

**Size / error table**

# Sketches

HyperLogLog

Theta

Quantiles

**Sketch Errors**

# Sketch errors

- Sketch errors are the **percentage difference** between the estimated value and true value.
- Sketch errors from error tables are **probabilistic**, not guarantees. You may see higher errors on individual measurements.
  - Example: 4.88% error at 99.73% confidence means there is a 99.73% chance that a randomly chosen measurement will have error below 4.88%. In the biz we refer to this as "probably approximately correct".
- Sketches **grow in size** very quickly as you dial up their accuracy. For this reason, we recommend that you stick near to the default accuracy.
- Similar to approximate ranking (top N), sketch errors are **magnified** when used in **non-additive measures**, especially for ratios with small denominators.
- All sketches provided by the DataSketches extension have *data-independent* error rates, and the provided error tables are also data-independent.


It's all part and parcel of the genie gig.


PHENOMENAL COSMIC POWERS!


Itty-bitty living space.

# Querying Data

**JSON over HTTP**

- ❏ Rest API based
- ❏ Queries and results are in {json] format
- ❏ Multiple query types e.g.
  - ❏ TopN
  - ❏ Group By
  - ❏ Select / Scan
  - ❏ Time bound
  - ❏ Segment(s) specific
  - ❏ Timeseries

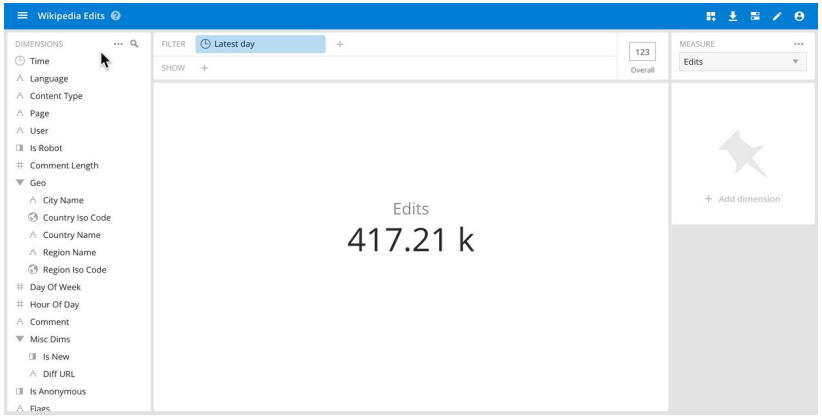
```
{
  "queryType": "groupBy",
  "dataSource": "sample_datasource",
  "granularity": "day",
  "dimensions": ["country", "device"],
  "limitSpec": { "type": "default", "l
  "filter": {
    "type": "and",
    "fields": [
```

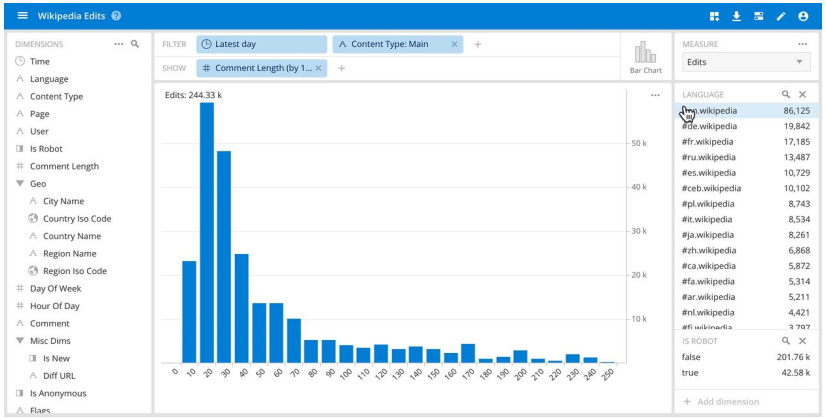
**In built SQL**

- Apache Calcite based parser
- Connect external BI tools using JDBC
- Run SQL via JSON over HTTP
- Supports approximate and lookup queries

```
SELECT COUNT(*)
FROM    Wikipedia
WHERE   page = 'Andre'
```

# The Goal


Wikipedia Edits
DIMENSIONS
Time
Language
Content Type
Page
User
Is Robot
Comment Length
Geo
City Name
Country Iso Code
Country Name
Region Name
Region Iso Code
Day Of Week
Hour Of Day
Comment
Misc Dims
Is New
Diff URL
Is Anonymous
FILTER
Latest day
SHOW
123
Overall
MEASURE
Edits
Edits
417.21 k
Add dimension


Wikipedia Edits
DIMENSIONS
Time
Language
Content Type
Page
User
Is Robot
Comment Length
Geo
City Name
Country Iso Code
Country Name
Region Name
Region Iso Code
Day Of Week
Hour Of Day
Comment
Misc Dims
Is New
Diff URL
Is Anonymous
FILTER
Latest day
Content Type: Main
SHOW
Comment Length (by 1...
Bar Chart
MEASURE
Edits
Edits: 244.33 k
LANGUAGE
en.wikipedia 86,125
de.wikipedia 19,842
fr.wikipedia 17,185
ru.wikipedia 13,487
es.wikipedia 10,729
ceb.wikipedia 10,102
pl.wikipedia 8,743
it.wikipedia 8,534
ja.wikipedia 8,261
zh.wikipedia 6,868
ca.wikipedia 5,872
fa.wikipedia 5,314
ar.wikipedia 5,211
nl.wikipedia 4,421
IS ROBOT
false 201.76 k
true 42.58 k
Add dimension

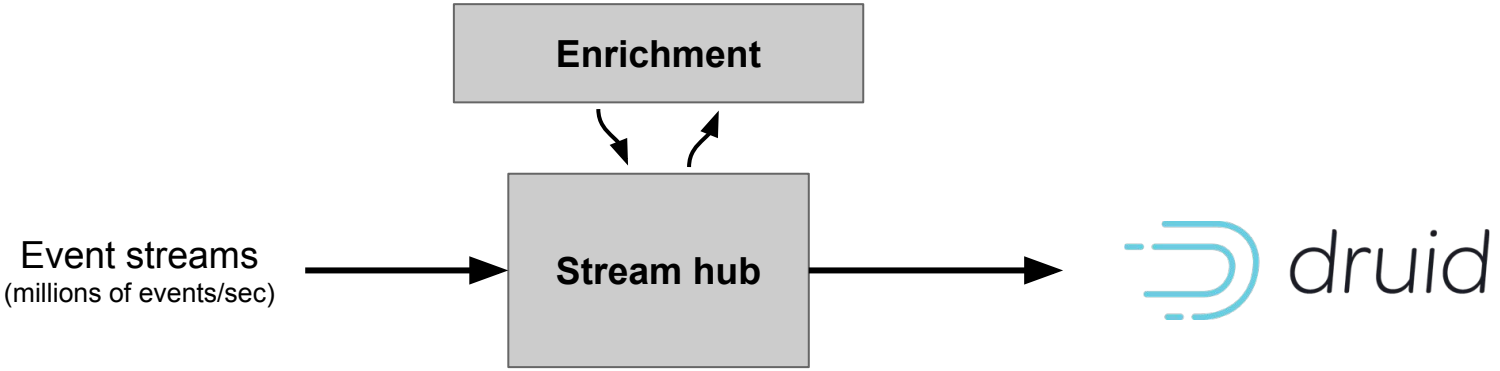

Wikipedia Edits
DIMENSIONS
Time
Language
Content Type
Page
User
Is Robot
Comment Length
Geo
City Name
Country Iso Code
Country Name
Region Name
Region Iso Code
Day Of Week
Hour Of Day
Comment
Misc Dims
Is New
Diff URL
Is Anonymous
FILTER
Latest 7 days
Content Type: Main
SHOW
Time (Hour)
Line Chart
MEASURE
Edits
Edits: 1.66 m
IS ROBOT
false 1,389.19 k
true 266.14 k
IS NEW
false 1,584.6 k
true 70.73 k
Add dimension
Mon 31
Tue 01
Wed 02
Thu 03
Fri 04
Sat 05
Sun 06

# Why this works

- Computers are fast these days
- Indexes help save work and cost
- But don’t be afraid to scan tables — it can be done efficiently


# Integration patterns

# Deployment patterns

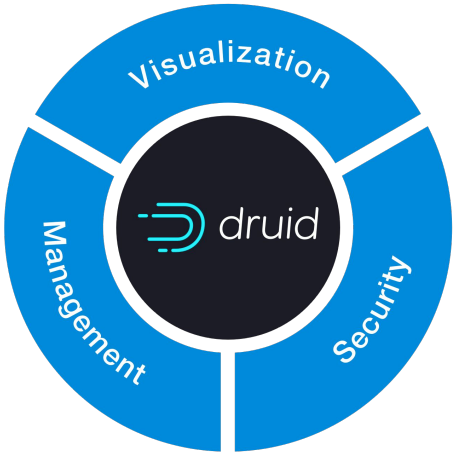



- Modern data architecture
- Centered around stream hub

# Who is Imply

Imply is a complete enterprise-ready solution built on Druid.

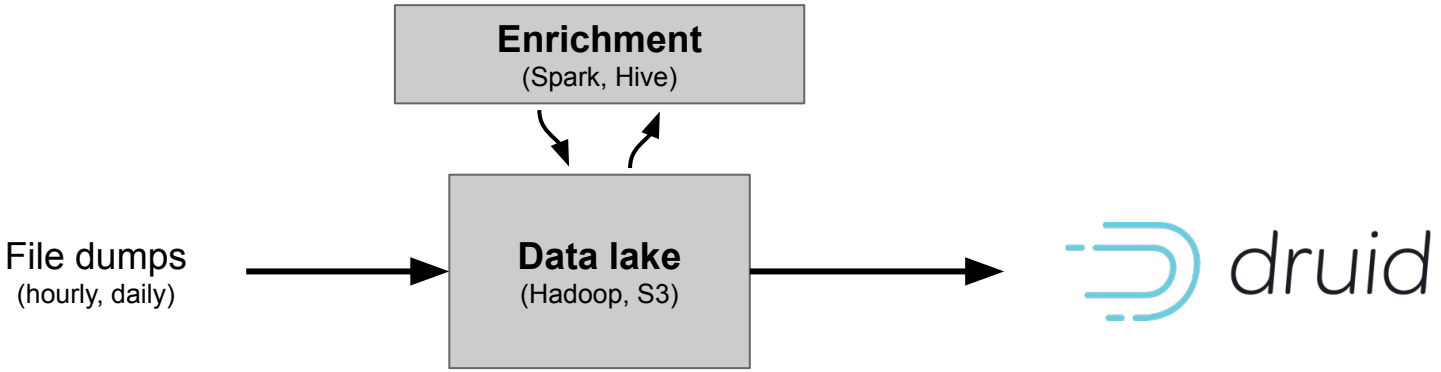
**Visualizations**
Simple data exploration
Point-and-click data explanation
Easy sharing and collaboration



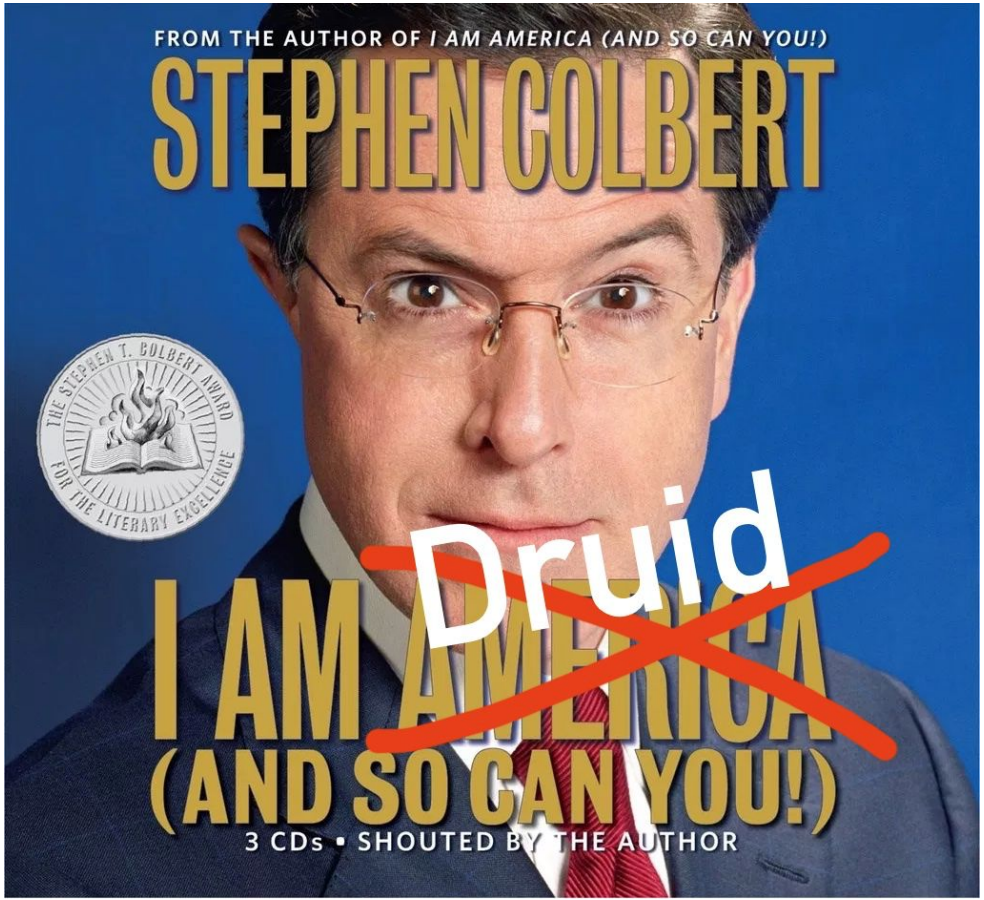

**Management and Operations**
Deployment and cluster manager
No downtime update functionality
Monitoring and alerting

**Security and Compliance**
User access control
Row and table level security
End-to-end encryption

# Deployment patterns




- (Slightly less) modern data architecture
- Centered around data lake


FROM THE AUTHOR OF I AM AMERICA (AND SO CAN YOU!)
STEPHEN COLBERT
THE STEPHEN T. COLBERT AWARD FOR THE LITERARY EXCELLENCE
Druid
I AM AMERICA
(AND SO CAN YOU!)
3 CDs • SHOUTED BY THE AUTHOR

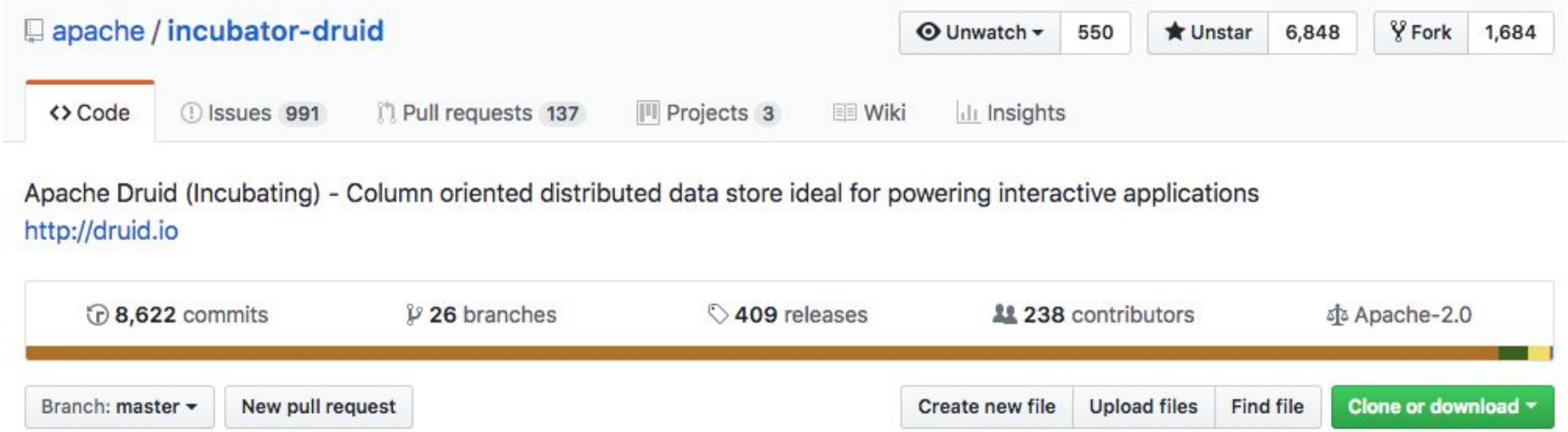
# Download

Apache Druid community site (new): https://druid.apache.org/

Apache Druid community site (legacy): http://druid.io/

Imply distribution: https://imply.io/get-started

# Contribute




https://github.com/apache/druid

# Stay in touch

Follow the Druid project on Twitter!

@druidio

Join the community!

http://druid.apache.org/