Updating Materialized Views and Caches Using Kafka

-or-

Why You Should Publish Data Changes to Kafka

Zach Cox

Prairie.Code() Oct 2016

https://github.com/zcox/twitter-microservices-example

About Me

- Building things with Apache Kafka since 2014
  - Currently at Uptake in Chicago: predictive analytics for industrial IoT
  - Previously at Banno in Des Moines: ad targeting for bank web sites
- Co-founded Pongr
  - Startup in Des Moines: photo marketing platform powered by messaging systems
- Software game since 1998
- Links
  - http://theza.ch
  - https://github.com/zcox
  - https://twitter.com/zcox
  - https://www.linkedin.com/in/zachcox

Remember These Things

1. Learn about Apache Kafka http://kafka.apache.org
2. Send events and data changes to Kafka
3. Denormalization is OK
4. **Up-to-date materialized views and caches**

Build a New Service

- Provides read access to data from many sources
  - Query multiple tables or databases
  - Complex joins, aggregations
- Response latency
  - 95% 5 msec
  - Max 10 msec
- Data update latency
  - 95% 1 sec
  - Max 10 sec

TWEETS	FOLLOWING	FOLLOWERS	LIKES	LISTS	MOMENTS
3,815	1,093	894	183	1	0

Zach Cox
@zcox

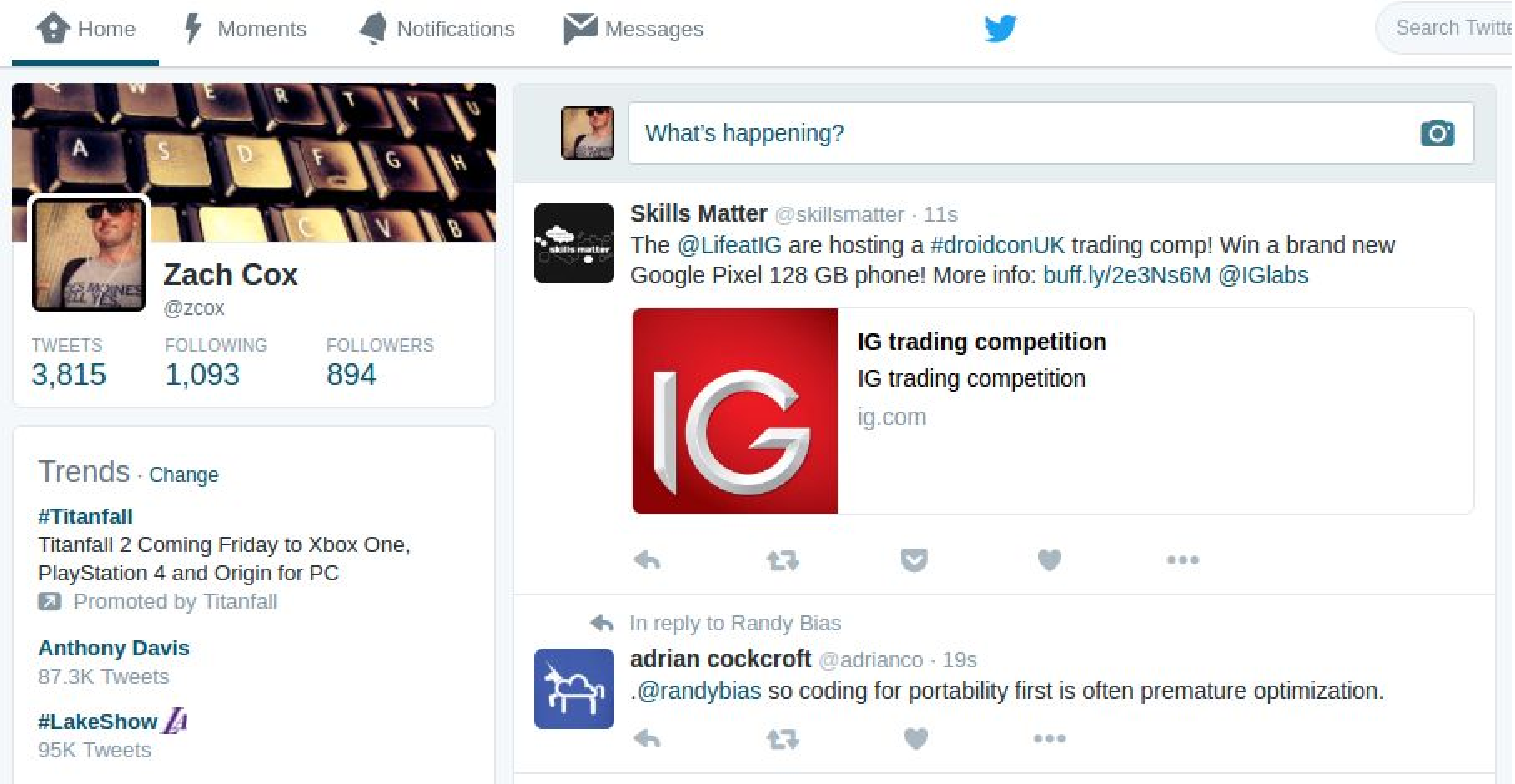
Engineering @Uptake

Chicago, IL
theza.ch
Joined January 2008

97 Photos and videos

Tweets   Tweets & replies   Media

Zach Cox @zcox · 13m
So unprofessional when conference speakers are still writing slides the day of their talk. Except when it's me, then it's totally fine...



Home
Moments
Notifications
Messages
Search Twitte
Zach Cox
@zcox
TWEETS
3,815
FOLLOWING
1,093
FOLLOWERS
894
Trends · Change
#Titanfall
Titanfall 2 Coming Friday to Xbox One, PlayStation 4 and Origin for PC
Promoted by Titanfall
Anthony Davis
87.3K Tweets
#LakeShow
95K Tweets
What's happening?
Skills Matter @skillsmatter · 11s
The @LifeatIG are hosting a #droidconUK trading comp! Win a brand new Google Pixel 128 GB phone! More info: buff.ly/2e3Ns6M @IGlabs
IG trading competition
IG trading competition
ig.com
In reply to Randy Bias
adrian cockcroft @adrianco · 19s
.@randybias so coding for portability first is often premature optimization.


43°
9:03
Edit profile
Zach Cox
@zcox
Engineering @Uptake
Chicago, IL
theza.ch
1,093 FOLLOWING
894 FOLLOWERS
TWEETS
MEDIA
LIKES
Zach Cox @zcox
13s
Replying to Kris Nuttycombe
Thanks for the inspiration! The content will at least be very fresh in my mind.
Zach Cox @zcox
11m

Response Times: The 3 Important Limits

by **JAKOB NIELSEN** on January 1, 1993

Topics: Applications Technology User Behavior Web Usability

Summary: There are 3 main time limits (which are determined by human perceptual abilities) to keep in mind when optimizing web and application performance.

Excerpt from Chapter 5 in my book Usability Engineering, from 1993:

The basic advice regarding response times has been about the same for thirty years [Miller 1968; Card et al. 1991]:

- **0.1 second** is about the limit for having the user feel that the system is **reacting instantaneously**, meaning that no special feedback is necessary except to display the result.
- **1.0 second** is about the limit for the **user's flow of thought** to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data.
- **10 seconds** is about the limit for **keeping the user's attention** focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect.

http://www.nngroup.com/articles/response-times-3-important-limits/

User Information Service

- One operation: get user information
- REST HTTP + JSON
- Input: userId
- GET /users/:userId

- Output:
  - userId
  - username
  - Name
  - Description
  - Location
  - Web page url
  - Joined date
  - Profile image url
  - Background image url
  - # tweets
  - # following
  - # followers
  - # likes
  - # lists
  - # moments

Existing RDBMS with Normalized Tables

- users
  - user_id
  - username
  - name
  - description
- tweets
  - tweet_id
  - text
  - user_id (FK users)
- follows
  - follow_id
  - follower_id (FK users)
  - followee_id (FK users)
- likes
  - like_id
  - user_id (FK users)
  - tweet_id (FK tweets)

Standard Solution: Query Existing Tables

- User fields
  - `SELECT * FROM users WHERE user_id = ?`
- # tweets
  - `SELECT COUNT(*) FROM tweets WHERE user_id = ?`
- # following
  - `SELECT COUNT(*) FROM follows WHERE follower_id = ?`
- # followers
  - `SELECT COUNT(*) FROM follows WHERE followee_id = ?`
- # likes
  - `SELECT COUNT(*) FROM likes WHERE user_id = ?`

Problems with Standard Solution

- Complex: multiple queries across multiple tables
- Potentially large aggregations at query time
  - Puts load on DB
  - Increases service response latency
  - Repeated on every query for same userId
- Shared data storage
  - Some other service writes to these tables (i.e. owns them)
  - When it changes schema, our service could break

Standard Solution: Add a Cache

- e.g. Redis
- Benefits
    - Faster key lookups than RDBMS queries
    - Store expensive computed values in cache and reuse them (i.e. **materialized view**)
- Usage
    - Read from cache first, if found then return cached data
    - Otherwise, read from DB, write to cache, return cached data

```scala
def getUser(id: String): User =
  readUserFromCache(id) match {
    case Some(user) => user
    case None =>
      val user = readUserFromDatabase(id)
      writeUserToCache(user)
      user
  }
```

```scala
def getUser(id: String): User =
  readUserFromCache(id) match {
    case Some(user) => user
    case None => //cache miss!
      val user = readUserFromDatabase(id)
      writeUserToCache(user)
      user
  }
```

```scala
def getUser(id: String): User =
  readUserFromCache(id) match {
    case Some(user) => user //stale?
    case None => //cache miss!
      val user = readUserFromDatabase(id)
      writeUserToCache(user)
      user
  }
```

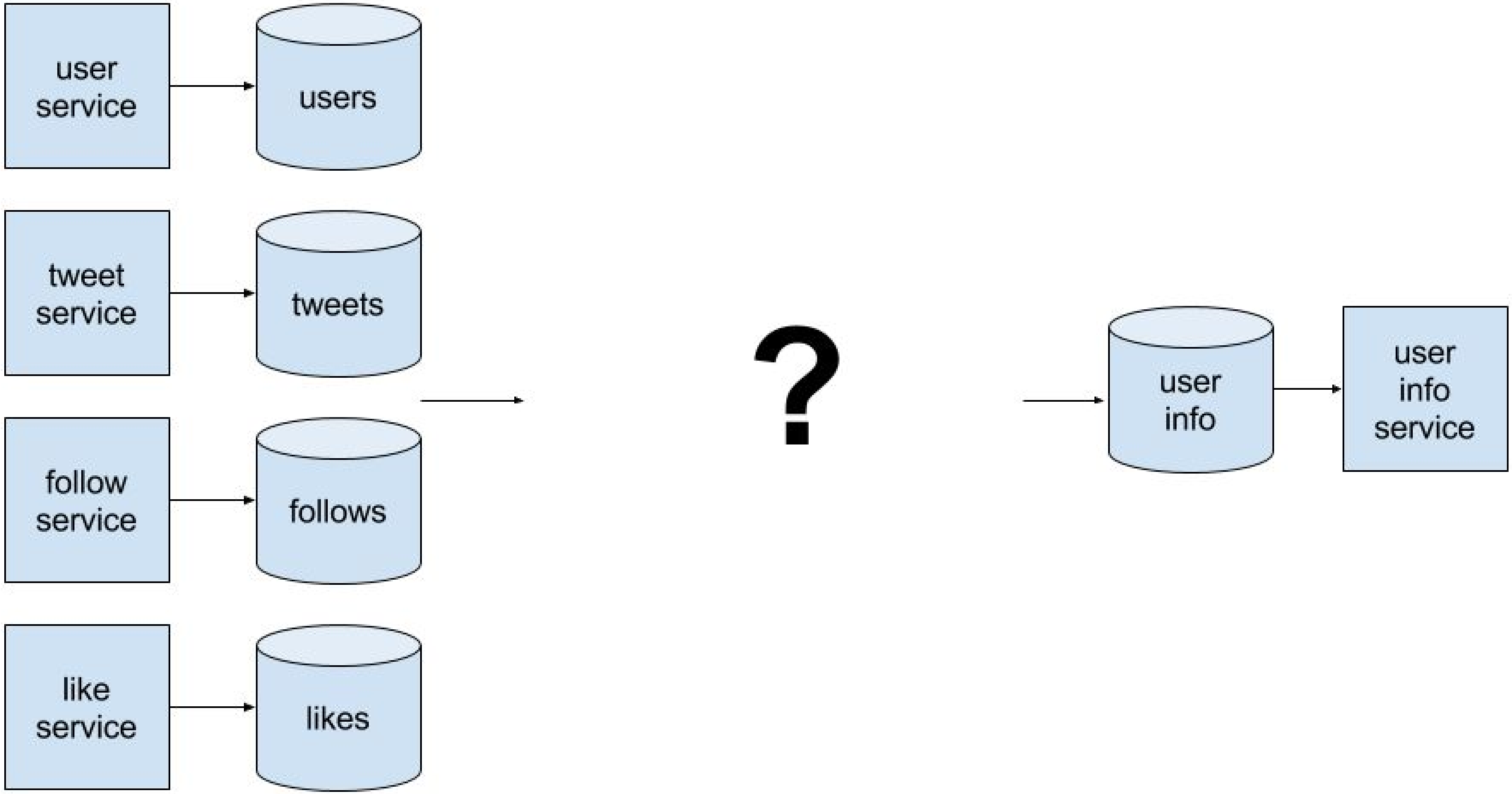
```scala
def getUser(id: String): User =
  readUserFromCache(id) match { //network latency
    case Some(user) => user //stale?
    case None => //cache miss!
      val user = readUserFromDatabase(id)
      writeUserToCache(user)
      user
  }
```

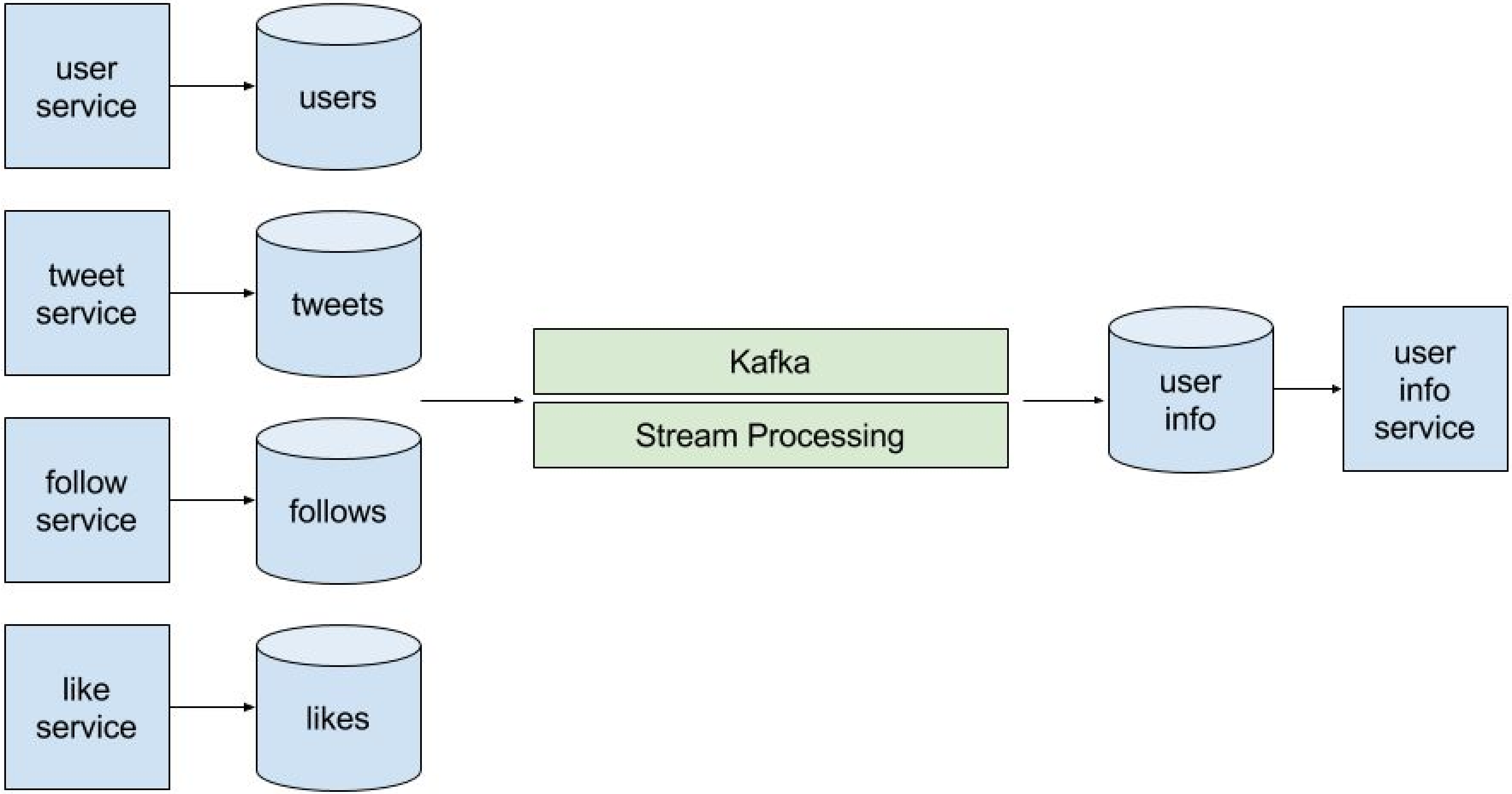
Problems with Standard Approach to Caches

- Operational complexity: someone has to manage Redis
- Code complexity: now querying two data stores and writing to one
- Cache misses: still putting some load on DB
- Stale data: cache is not updated when data changes
- Network latency: cache is remote

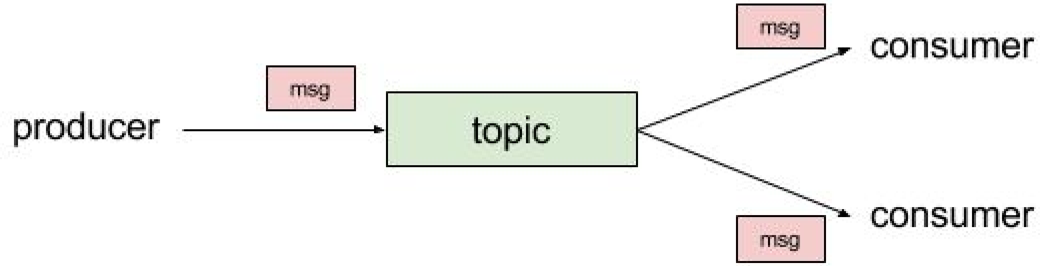
Can We Solve These Problems?

- Yes: **If cache is always updated**
- Complexity: only read from cache
- Cache misses: cache always has *all* data
- Stale data: cache always has *updated* data
- Network latency: if cache is local to service (bonus)


user service
users
tweet service
tweets
follow service
follows
like service
likes
?
user info
user info service


user service
users
tweet service
tweets
follow service
follows
like service
likes
Kafka
Stream Processing
user info
user info service

Kafka: Topics, Producers, Consumers



- Horizontally scalable, durable, highly available, high throughput, low latency

Kafka: Messages

- Message is a (key, value) pair
- Key and value are byte arrays (BYO serialization)
- Key is typically an ID (e.g. userId)
- Value is some payload (e.g. page view event, user data updated)

Kafka: Producer API

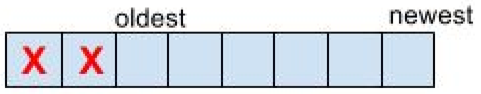
```
val props = … //kafka host:port, other configs

val producer = new KafkaProducer[K, V](props)

producer.send(topic, key, value)
```

Kafka: Consumer API

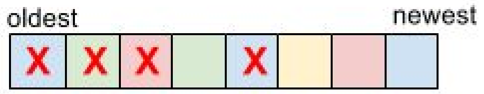
```
val props = … //kafka host:port, other configs

val consumer = new KafkaConsumer[K, V](props)

consumer.subscribe(topics)

while (true) {

  val messages = consumer.poll(timeout)

  //process list of messages

}
```

Kafka: Types of Topics

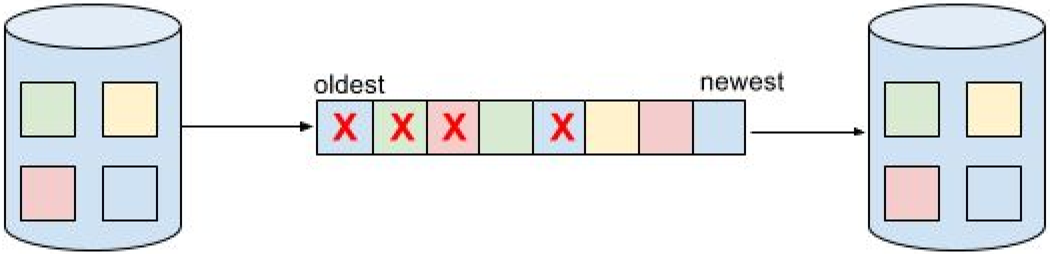
- Record topic
  - Finite topic retention period (e.g. 7 days)
  - Good for user activity, logs, metrics



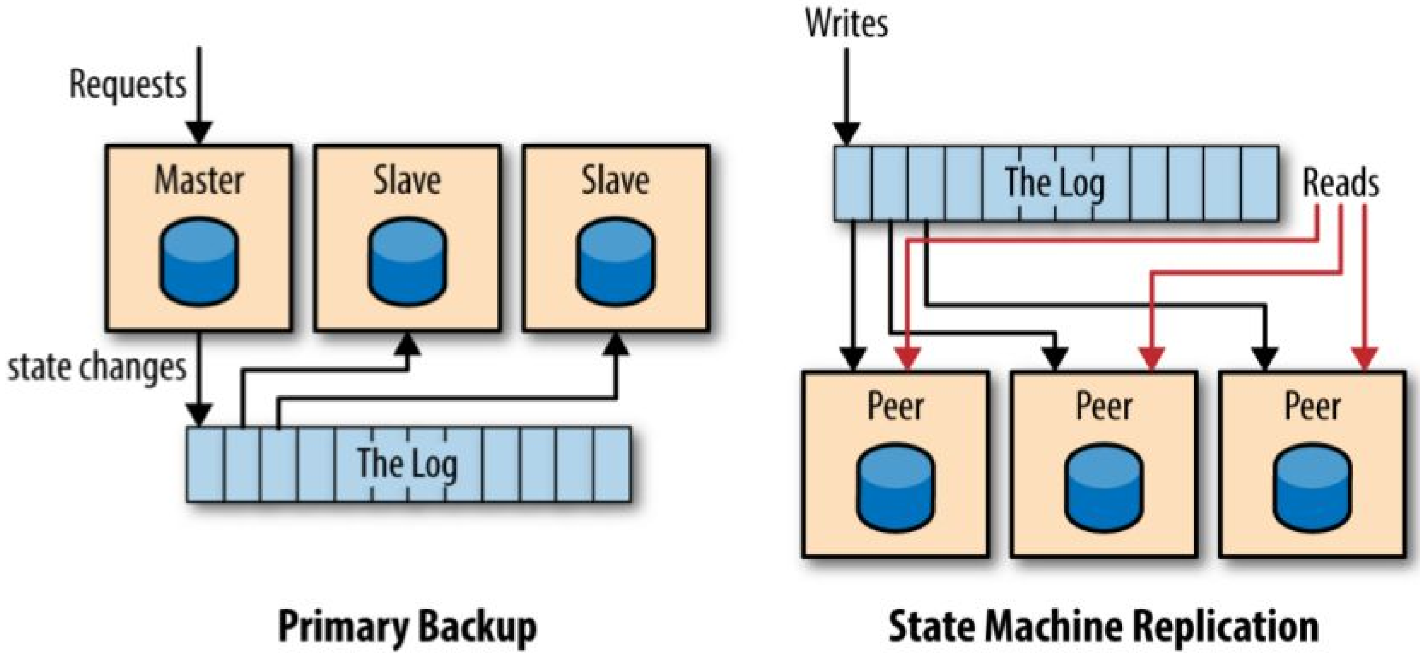
- Changelog topic
  - Log-compacted topic: retains newest message for each key
  - Good for entities/table data



Kafka: Tables and Changelogs are Dual



Database Replication



Credit: I Heart Logs http://shop.oreilly.com/product/0636920034339.do

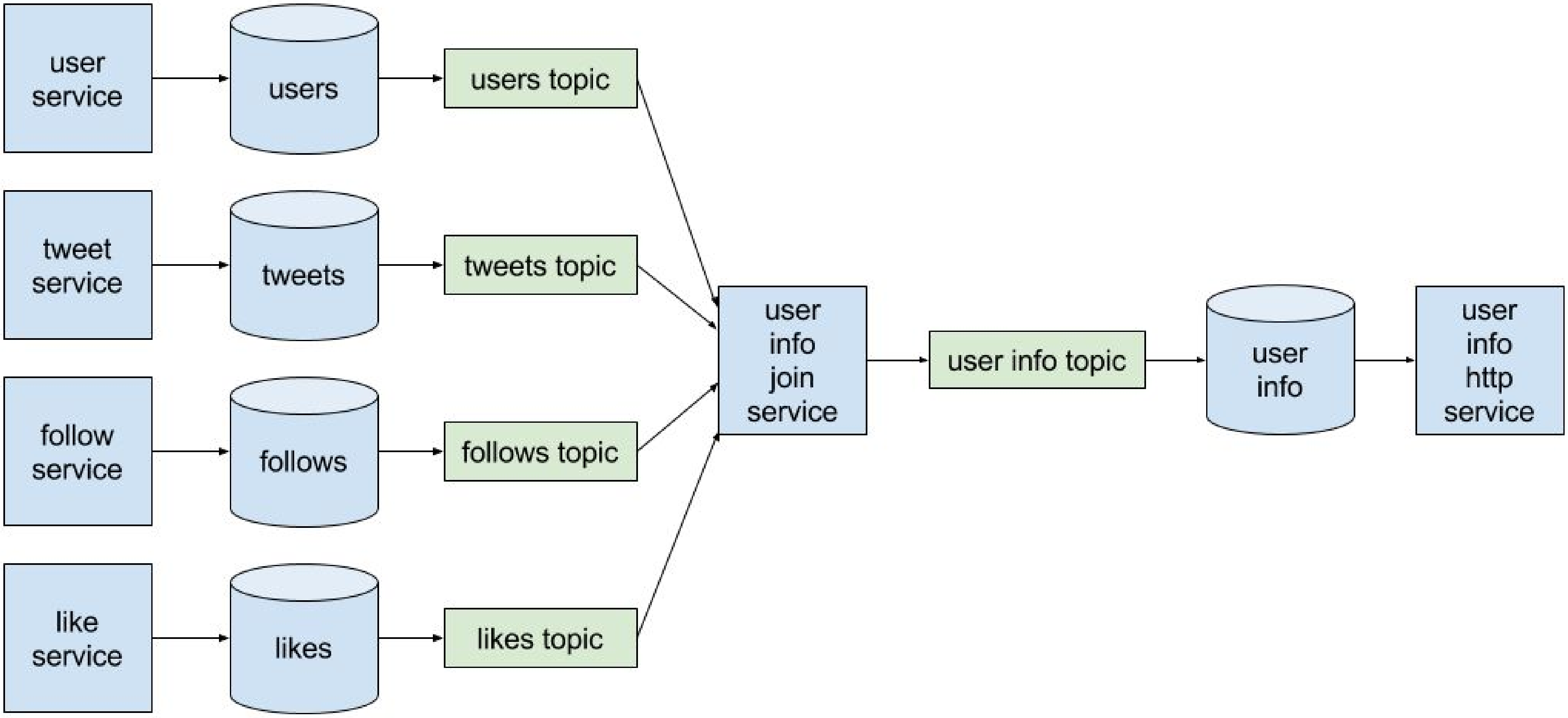
DB to Kafka

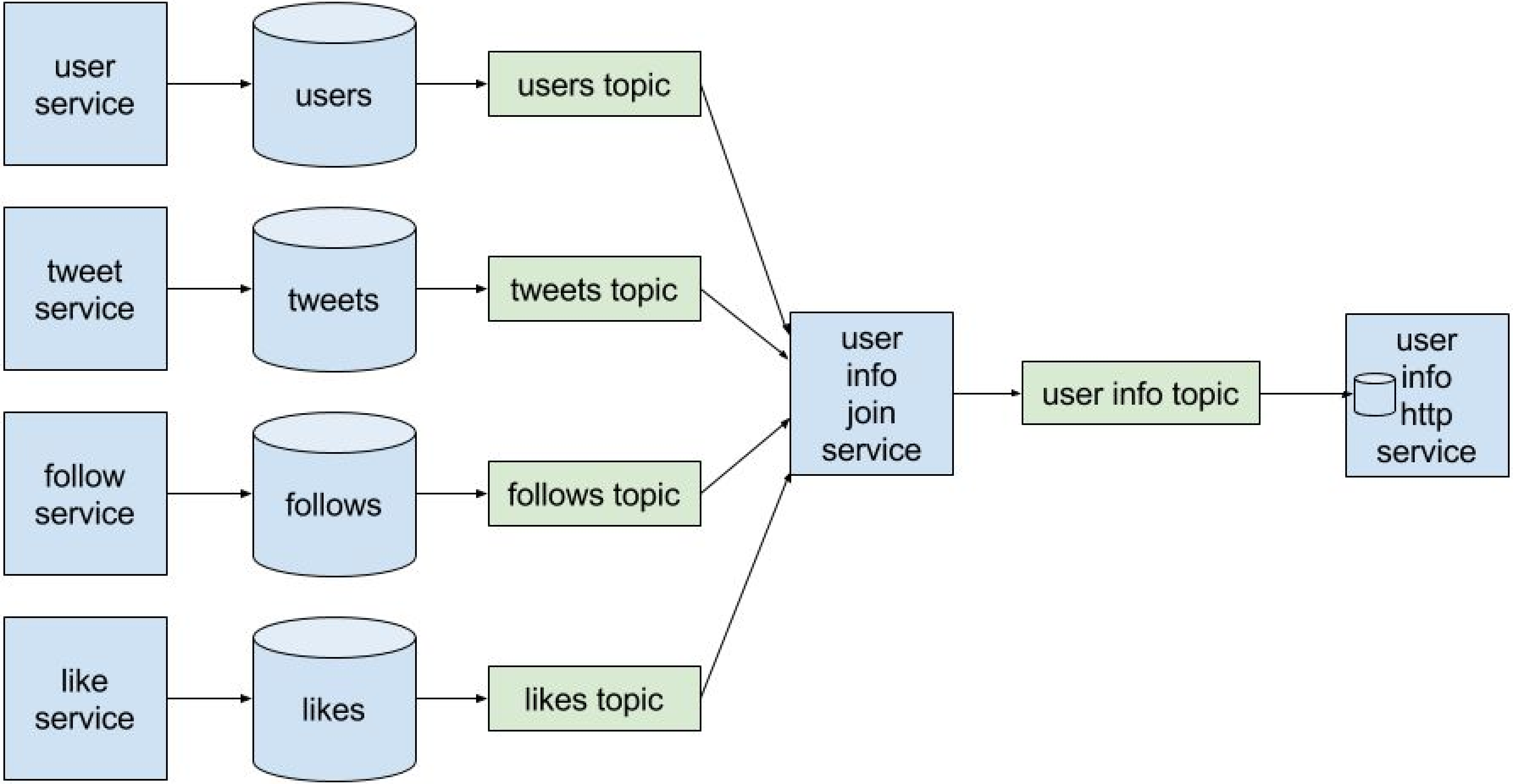
- Change data capture
  - Kafka Connect http://kafka.apache.org/documentation#connect
  - Bottled Water https://github.com/confluentinc/bottledwater-pg
- Dual writes
  - Application writes to both DB and Kafka
  - Prefer CDC

Kafka Streams

- Higher-level API than producers and consumers
- Just a library (no Hadoop/Spark/Flink cluster to maintain)

```
val tweetCountsByUserId = builder.stream(tweetsTopic)

    .selectKey((tweetId, tweet) => tweet.userId)

    .countByKey("tweetCountsByUserId")

val userInformation = builder.table(usersTopic)

    .leftJoin(tweetCountsByUserId,

              (user, count) => new UserInformation(user, count))

userInformation.to(userInformationTopic)
```

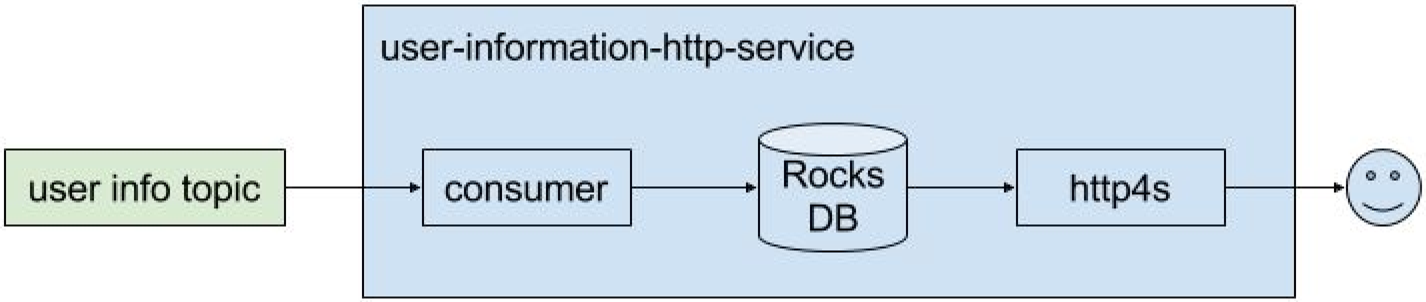

user service
users
users topic
tweet service
tweets
tweets topic
follow service
follows
follows topic
like service
likes
likes topic
user info join service
user info topic
user info
user info http service


user service
users
users topic
tweet service
tweets
tweets topic
follow service
follows
follows topic
like service
likes
likes topic
user info join service
user info topic
user info http service

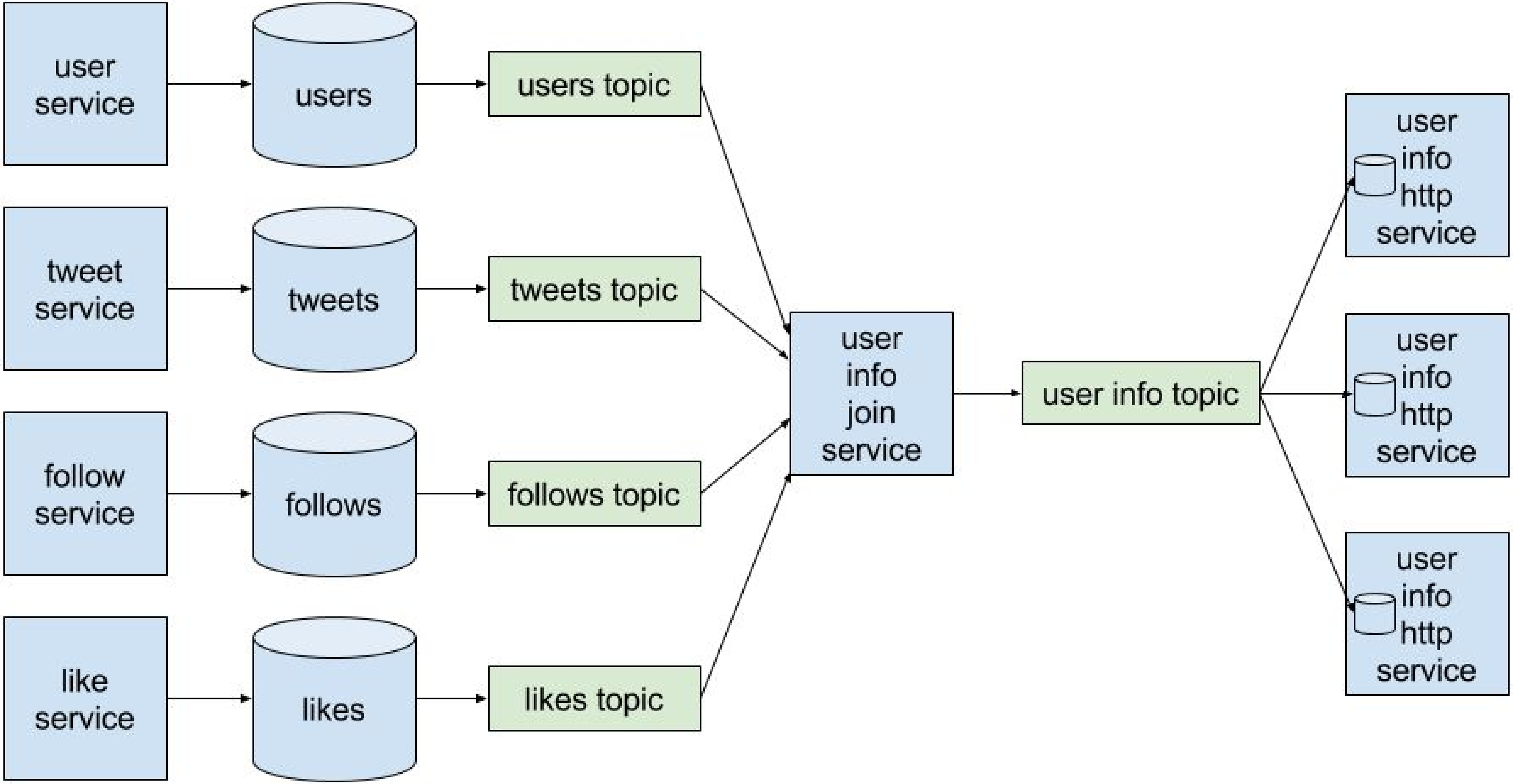
RocksDB

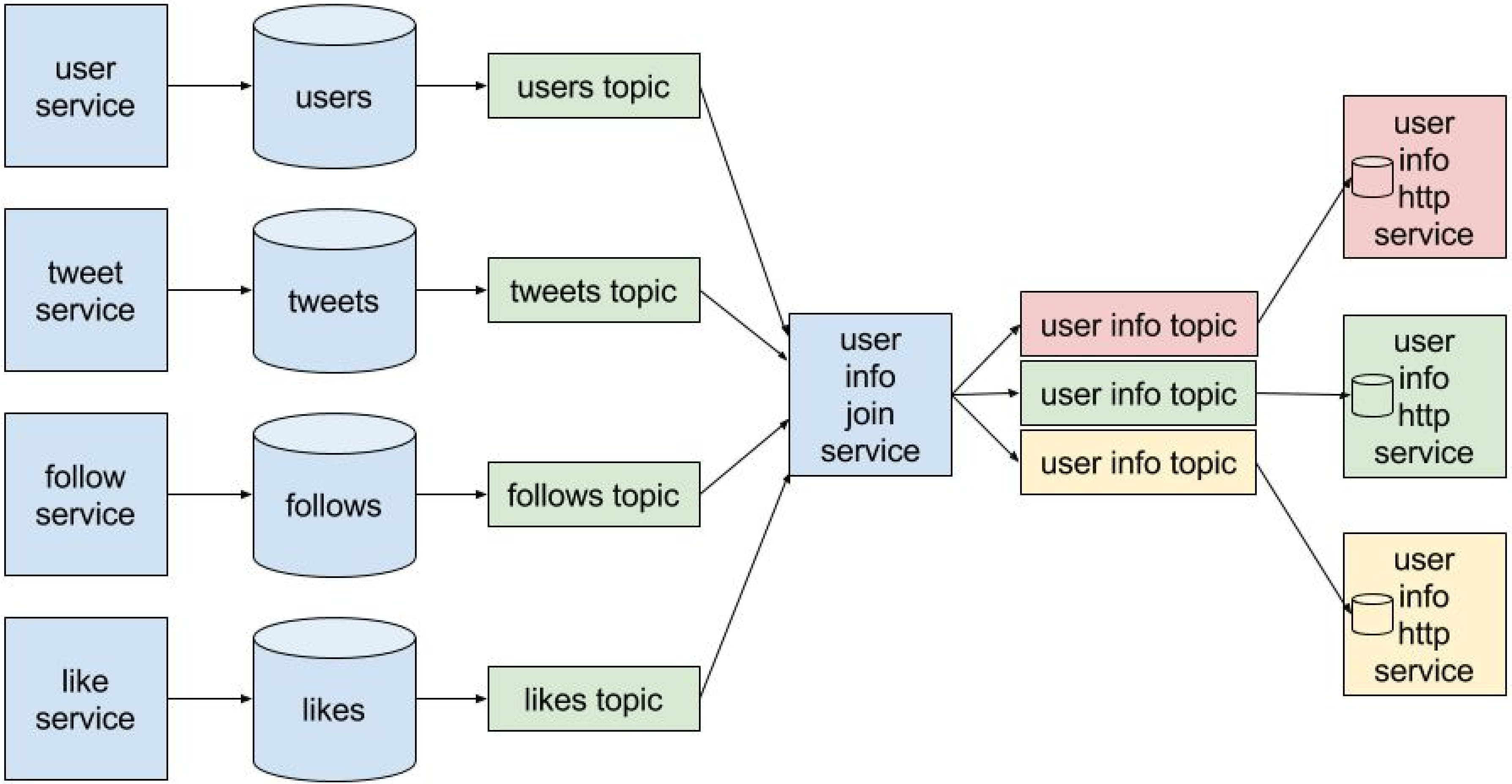
- Key-value store
- In-process
  - Local (not remote)
  - Library (not a daemon/server)
- Mostly in-memory, spills to local disk
  - Usually an under-utilized resource on app servers
  - 100s of GBs? TBs?
  - AWS EBS 100GB SSD $10/mo
- http://rocksdb.org

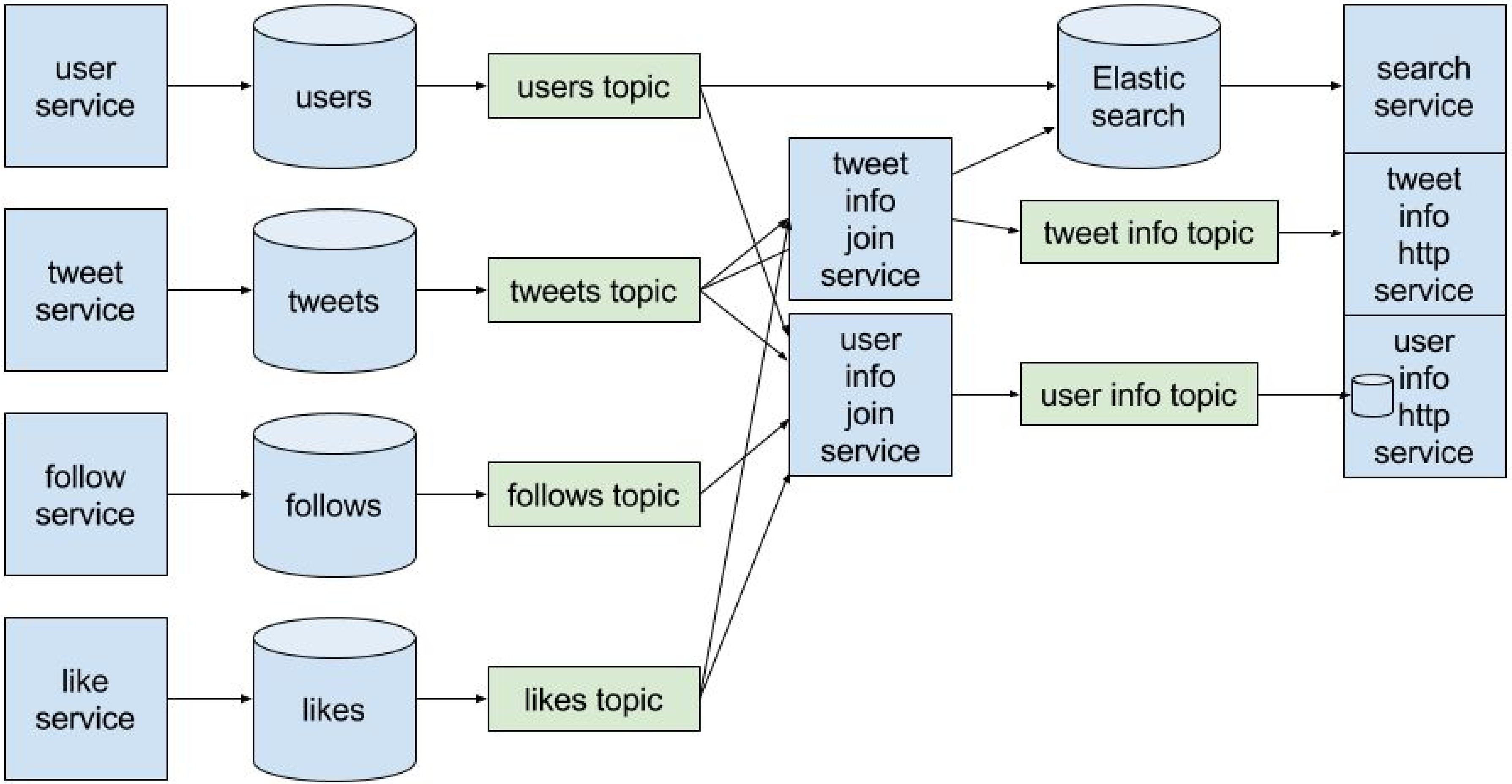
HTTP Service Internals

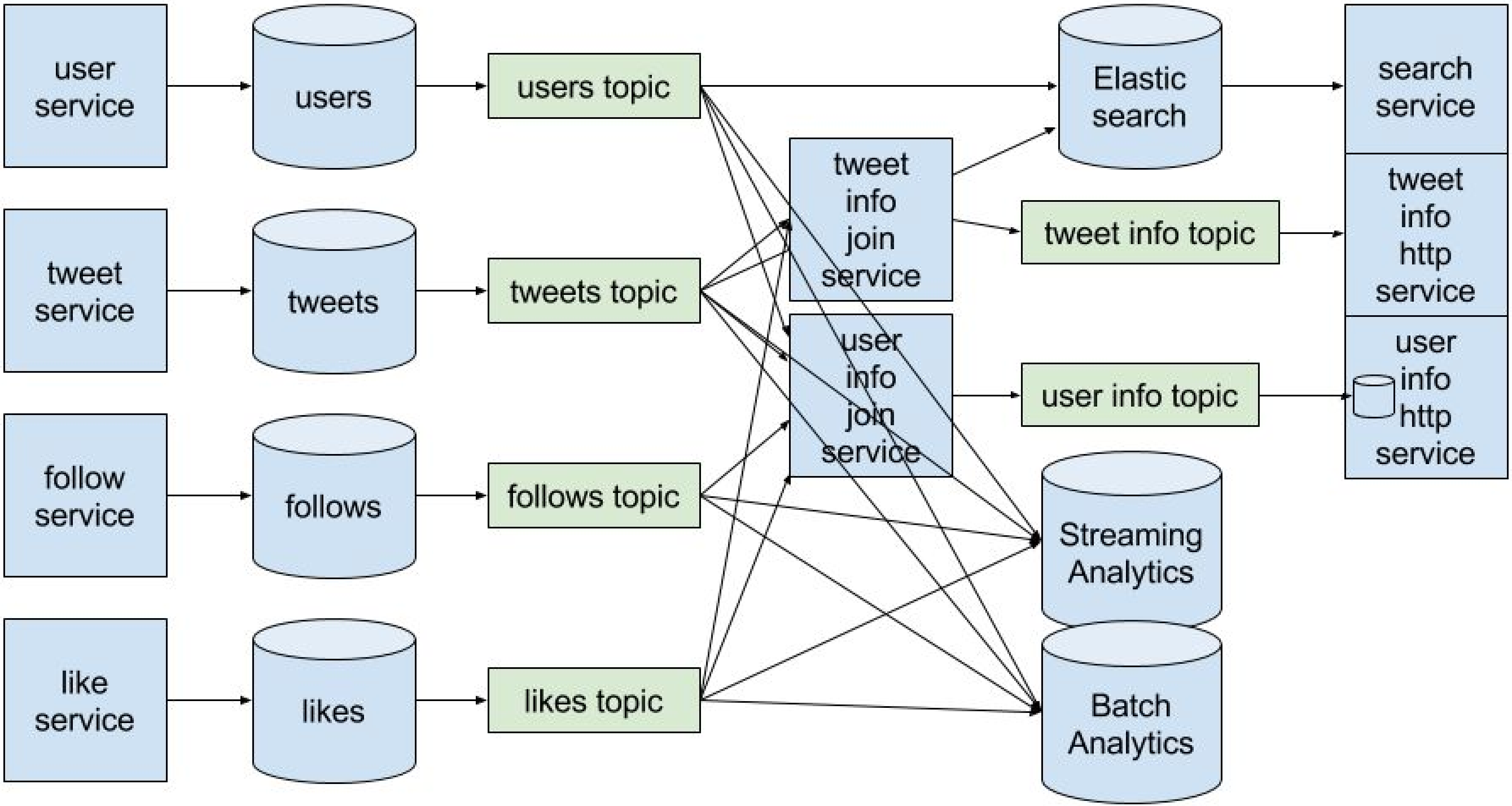


Live Demo!


user service
users
users topic
tweet service
tweets
tweets topic
follow service
follows
follows topic
like service
likes
likes topic
user info join service
user info topic
user info http service
user info http service
user info http service


user service
users
users topic
tweet service
tweets
tweets topic
follow service
follows
follows topic
like service
likes
likes topic
user info join service
user info topic
user info topic
user info topic
user info http service
user info http service
user info http service


user service
users
users topic
Elastic search
search service
tweet service
tweets
tweets topic
tweet info join service
tweet info topic
tweet info http service
user info join service
user info topic
user info http service
follow service
follows
follows topic
like service
likes
likes topic

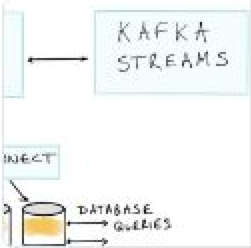

user service
users
users topic
Elastic search
search service
tweet service
tweets
tweets topic
tweet info join service
tweet info topic
tweet info http service
user info join service
user info topic
user info http service
follow service
follows
follows topic
Streaming Analytics
like service
likes
likes topic
Batch Analytics

Kafka Streams Interactive Queries

Jay Kreps
@jaykreps

Following

1. What does it mean to have "interactive query" support for Kafka's Streams API? The long story is in the blog:



Unifying Stream Processing and Interactive Queries in Apa...
We are excited to announce Interactive Queries, a new feature for stream processing with Apache Kafka™.
confluent.io

RETWEETS 32 LIKES 64

6:21 PM - 26 Oct 2016

http://www.confluent.io/blog/unifying-stream-processing-and-interactive-queries-in-apache-kafka/

Confluent Schema Registry

- Serialize messages in Kafka topics using Avro
- Avro schemas registered with central server
- Anyone can safely consume data in topics
- http://docs.confluent.io/3.0.1/schema-registry/docs/index.html