

DATA SERVICES AT RACKSPACE SERIES

Pre-aggregated analytics and social feeds using MongoDB



JON HYMAN
Co-Founder and CIO

Appboy



GREG AVOLA
CTO, Co-Founder
and Developer

Untappd



KENNY GORMAN
Founder of ObjectRocket
Chief Architect

Rackspace



J.R. ARREDONDO
Director, Product Marketing

Rackspace



Data Services at Rackspace



2 acquisitions for
MongoDB and Redis apps



REDIS TO GO

2 offerings in partnership
with Hortonworks for Hadoop-based
applications



Strong portfolio of
traditional offerings



The right tool for the right job

Relational

**Data
Integrity**

SQL



Documents

**Flexible
Schema**

Scale



Key-value

**Fast
Retrieval**

**Data
structures**

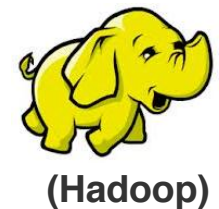
REDIS TO GO



Distributed large sets

**Distributed
Processing**

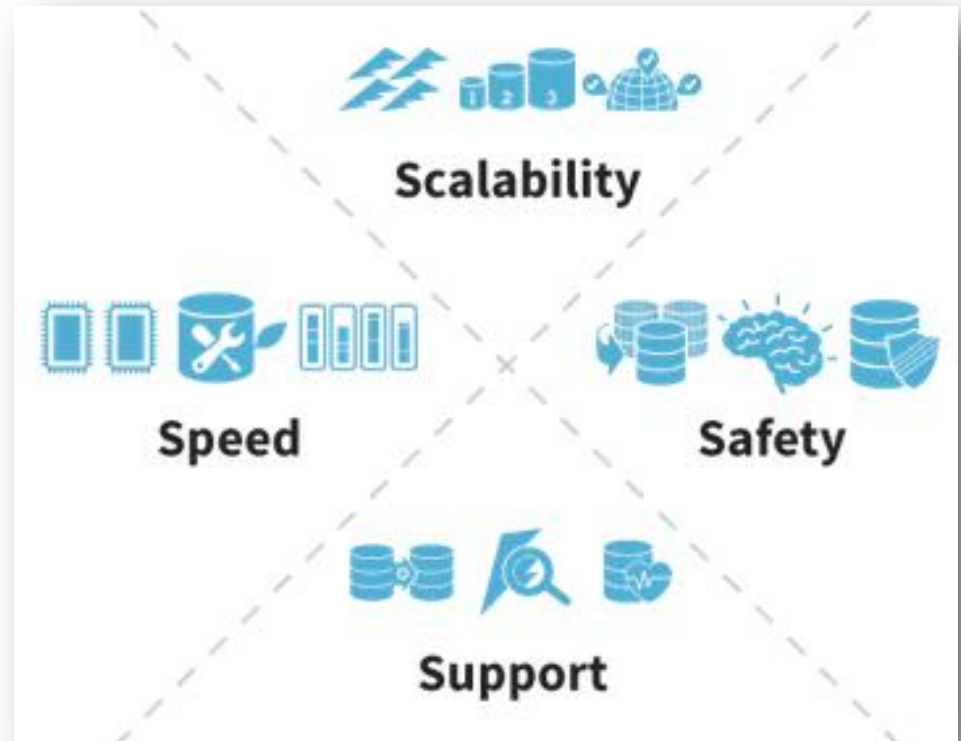
Big Data



What is ObjectRocket?

Think about 4 “S”

A **fully managed MongoDB service** that is purpose-built to be highly available, automatically sharded, and very fast.



ObjectRocket high level summary

Think about 4 “S”

SCALABILITY

Automatically sharded (all you have to do is select a shard key)

Multiple plans, starting at 1GB with no sharding to custom plans

Choice of regions: US East, US West, London (Sydney and Hong Kong coming up)

AWS Direct Connect or Rackspace ServiceNet

SPEED

Purpose-built design **optimized for MongoDB**

Fusion-IO storage for high throughput

Container-based architecture

SAFETY

Smart provisioning to reduce the possibility of downtime

Redundancy and automatic backups

Security: ACLs, MongoDB authentication, SSL termination, encrypted replication links

SUPPORT

Deep MongoDB expertise

Monitoring

Migration Services

Who is speaking to you today?



Jon Hyman (Appboy)

Appboy addresses the rising need of apps to keep audiences engaged. By bringing data-driven marketing automation, segmentation, multi-channel messaging and mobile marketing experts together, Appboy has become the pioneer of Mobile Relationship Management. Co-Founder and CIO Jon Hyman is responsible for the infrastructure behind the tools designed for app developers to segment users by behavior and deliver personalized, relevant content to them via multi-channel messaging, helping to increase engagement and understanding within the app.



Greg Avola (Untappd)

Untappd is a new way to socially share and explore the world of beer with your friends and the world. Living in the craft beer haven of New York City, Greg is the backend developer for Untappd. After experiencing Rare Vos for the first time, he instantly fell in love with craft beer. While some people enjoy reading books or watching movies, Greg's passion is to code. That being said, after Tim and Greg came up with the idea of Untappd, Greg had a working prototype the next day. Being able to combine his passion for development and craft beer allowed Untappd to be born.

Appboy and MongoDB

Jon Hyman

MongoDB & ObjectRocket Webinar, Feb. 12, 2014



@appboy

@jon_hyman

A LITTLE BIT ABOUT US & APPBOY

(who we are and what we do)



Jon Hyman
CIO :: @jon_hyman
Harvard
Bridgew
ater



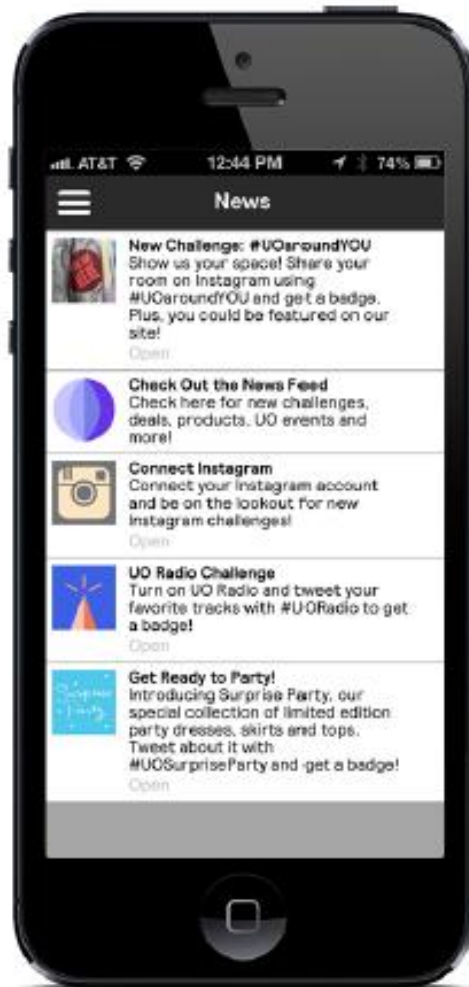
Appboy is an **app marketing
automation** platform for apps

■ Appboy improves engagement by helping you understand your app users

- **IDENTIFY** - Understand demographics, social and behavioral data
- **SEGMENT** - Organize customers into groups based on behaviors, events, user attributes, and location
- **ENGAGE** - Message users through push notifications, emails, and multiple forms of in-app messages



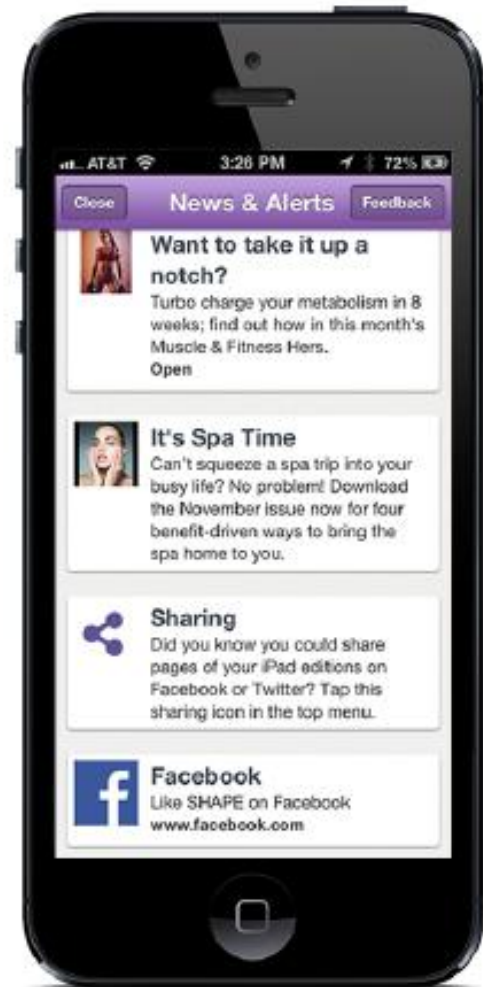
Use Case: Customer engagement begins with onboarding



Urban Outfitters



textPlus



Shape Magazine

Agenda

- How flexible schemas make storing arbitrary data super easy
- How to quickly store time series data in MongoDB using flexible schemas
- Learn how flexible schemas can easily provide breakdowns across dimensions



Flexible Schemas, Part 1:

EXTENSIBLE USER PROFILES

Extensible User Profiles

Appboy creates a rich user profile on every user who opens one of our customers' apps



Jovan Quigley

✉ jovan.quigley+demo@appboy.com

Overview | Engagement | Feedback | Attributes & Events | Social

Profile


Gender	Female
Age Group	21 and under
Country	DE

App Usage

Sessions	24
Median Session Length	33 seconds
Last Used	5 days ago
First Used	14 days ago
Money Spent In-App	\$0.00
Push Notification Enabled	No

Extensible User Profiles

We also let our customers add their own custom attributes



Jovan Quigley
✉ jovan.quigley+demo@appboy.com

[Overview](#) | [Engagement](#) | [Feedback](#) | **[Attributes & Events](#)** | [Social](#)


Custom Attributes

Attribute Name	Value
has_avatar	true
highest_score	1000
last_beat_level	6
visited_website	false

Extensible User Profiles

Let's talk schema

```
{  
  first_name: "Jovan",  
  email: "jovan+demo@appboy.com",  
  dob: 1994-10-24,  
  gender: "F",  
  country: "DE",  
  ...  
}
```



Jovan Quigley
✉ jovan.quigley+demo@appboy.com

[Overview](#) | [Engagement](#) | [Feedback](#) | [Attributes & Events](#) | [Social](#)

Profile

Gender	Female
Age Group	21 and under
Country	DE


App Usage

Sessions	24
Median Session Length	33 seconds
Last Used	5 days ago
First Used	14 days ago
Money Spent In-App	\$0.00
Push Notification Enabled	No

Extensible User Profiles

Custom attributes can go alongside other fields!

```
{
  first_name: "Jovan",
  email: "jovan+demo@appboy.com",
  dob: 1994-10-24,
  gender: "F",
  custom: {
    has_avatar: true,
    highest_score: 1000,
    visited_website: false,
    ...
  },
  ...
}
```



Jovan Quigley
✉ jovan.quigley+demo@appboy.com

Overview | Engagement | Feedback | **Attributes & Events** | Social

Custom Attributes

Attribute Name	Value
has_avatar	true
highest_score	1000
last_beat_level	6
visited_website	false

```
db.users.find(...).update({$set: {"custom.has_avatar":true}})
```


Extensible User Profiles

- **Pros**

- Easily extensible to add any number of fields
- Don't need to worry about type (bool, string, integer, float, etc.): MongoDB handles it all
- Can do atomic operations like \$inc easily
- Easily queryable, no need to do complicated joins against the right value column

- **Cons**

- Can take up a lot of space “this_is_my_really_long_custom_attribute_name_weeeeeee”
- Can end up with mismatched types across documents

{ visited_website: true }

{ visited_website: “yes” }

Extensible User Profiles - How to Improve the Cons

Space Concern

Tokenize values, use a field map:

```
{
  first_name: "Jovan",
  email: "jovan+demo@appboy.com",
  dob: 1994-10-24,
  gender: "F",
  custom: {
    0: true,
    1: 1000,
    2: false,
    ...
  },
  ...
}
```

```
{
  has_avatar: 0,
  highest_score: 1,
  visited_website: 2
}
```

You should also limit the length of values

Extensible User Profiles - How to Improve the Cons

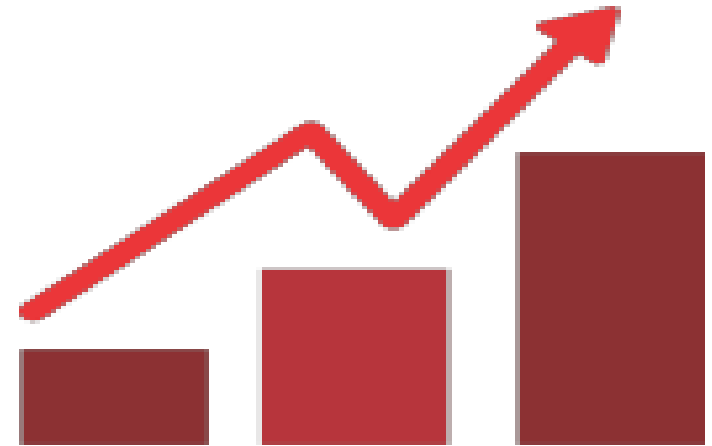
Type Constraints

Handle in the client, store expected types in a map and coerce/reject bad values

```
{  
  has_avatar: Boolean,  
  highest_score: Integer,  
  favorite_color: String  
}
```

Extensible User Profiles Summary

- MongoDB is probably the best tool you can use for custom attributes
- Just as simple to CRUD as any other field on the document
- Be sure to handle space and type concerns





Flexible Schemas, Part 2:

PRE-AGGREGATED ANALYTICS

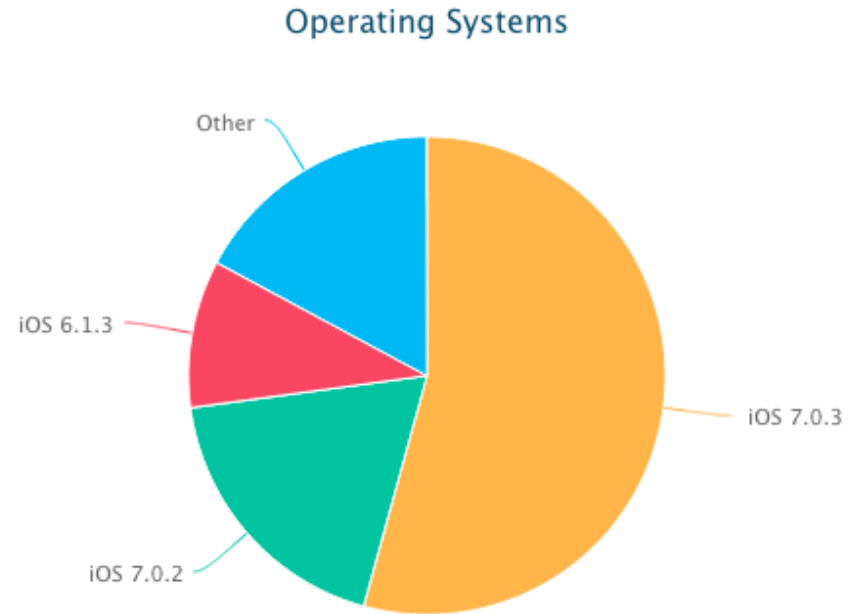
What kinds of analytics does Appboy track?

- **Lots of time series data**
 - App opens over time
 - Events over time
 - Revenue over time
 - Marketing campaign stats and efficacy over time



What kinds of analytics does Appboy track?

- **Breakdowns***
 - Device types
 - Device OS versions
 - Screen resolutions
 - Revenue by product



* We also care about this over time!

Typical time series collection

Log a new row for each open received

```
{  
  timestamp: 2013-11-14 00:00:00 UTC,  
  app_id: App identifier  
}
```

```
db.app_opens.find({app_id: A, timestamp: {$gte: date}})
```

Pro: Really, really simple. Easy to add attribution to users.

Con: You need to aggregate the data before drawing the chart; lots of documents read into memory, lots of dirty pages

Fewer documents with pre-aggregation iteration 1

Create a document that groups by the time period

```
{  
  app_id: App identifier,  
  date: Date of the document,  
  hour: 0-23 based hour this document represents,  
  opens: Number of opens this hour  
}
```

```
db.app_opens.update({date: D, app_id: A, hour: 0}, {$inc: {opens:1}})
```

Pro: Really easy to draw histograms

Con: We never care about an hour by itself. We lose attribution.

Fewer documents with pre-aggregation iteration 2

Create a document by day and have each hour be a field

```
{  
  app_id: App identifier,  
  date: Date of the document,  
  total_opens: Total number of opens this day,  
  0: Number of opens at midnight,  
  1: Number of opens at 1am,  
  ...  
  23: Number of opens at 11pm  
}
```

```
db.app_opens.update(  
  {date: D, app_id: A},  
  {$inc: {"0":1, total:1}}  
)
```

Pro: Document count is low, easy to use aggregation framework for longer spans, fast: document should be in working set

Fewer documents with pre-aggregation iteration 2

- **What about looking at different dimensions?**
 - App opens by device type (e.g., how do iPads compare to iPhones?)
 - Demographics (gender, age group)



Solution!

FLEXIBLE SCHEMAS!

Fewer documents with pre-aggregation iteration 3

Dynamically add dimensions in the document

```
{
  app_id: App identifier,
  date: Date of the document,
  totals: {
    app_opens: Total number of opens this day,
    devices: {
      "iPad Air": Total number of opens on the iPad Air,
      "iPhone 4": Total number of opens on the iPhone 4,
    },
    genders: {
      male: Total number of opens from male users,
      female: Total number of opens from female users
    },
    ...
  },
  0: {
    app_opens: Number of opens at midnight,
    devices: {
      "iPad Air": Number of opens on the iPad Air at midnight,
      "iPhone 4": Number of opens on the iPhone 4 at midnight,
    },
    ...
  },
  ...
}

db.app_opens.update({date: D, app_id: A}, {$inc: {"0":1, total:1}})
```

Pre-aggregated analytics

- **Pros**

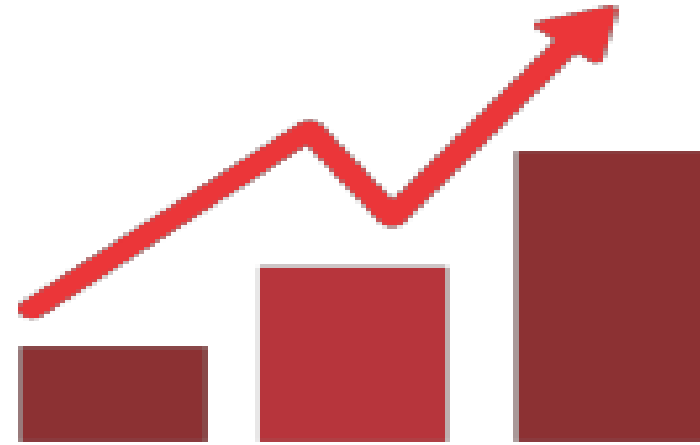
- Easily extensible to add other dimensions
- Still only using one document, therefore you can create charts very quickly
- You get breakdowns over a time period for free

- **Cons**

- Pre-aggregated data has no attribution
- Have to know questions ahead of time
- Not infinitely scaleable
- Timezones are awful problems to deal with

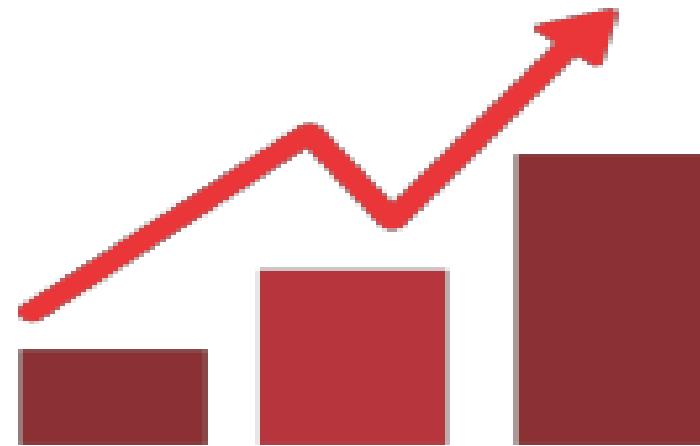
Pre-aggregated analytics summary

- Get started tracking time series data quickly
- You get breakdowns for free
- Adding dimensions is super simple
- No attribution, need to know questions ahead of time
- Don't just rely on pre-aggregated analytics



Working with ObjectRocket

- Knowledgable, MongoDB experts
- Helped us scale to many billions of data points each month
- Support is top-notch, response times are wicked fast (and they stay on top of MongoDB, Inc. when tickets get escalated)
- Great for cost: 3 full nodes (not 2 and 1 arbiter), backups included, good step-wise pricing model as you add more shards



Thanks! Questions?

jon@appboy.com



@appboy

@jon_hyman

Experiences with MySQL and MongoDB in a social app

Who am I?



Live in NYC

Day-time Job: Web Developer
at ABC News (US)

Night-time Job: CTO, Co-
Founder of Untappd

Loves: Data, Beer and
Javascript



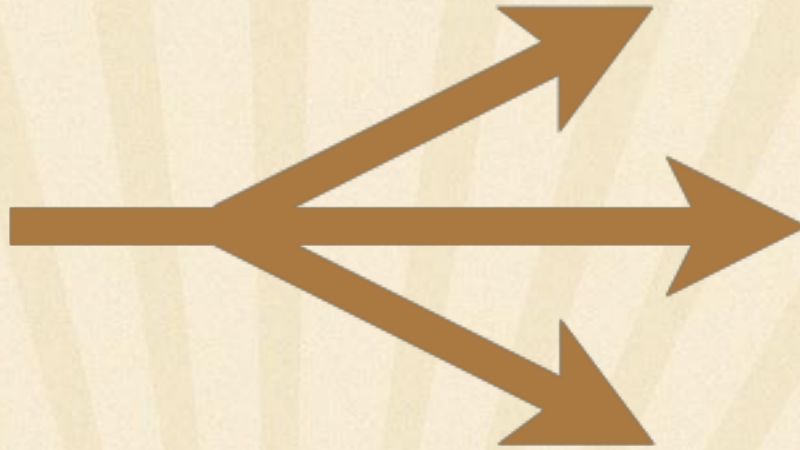
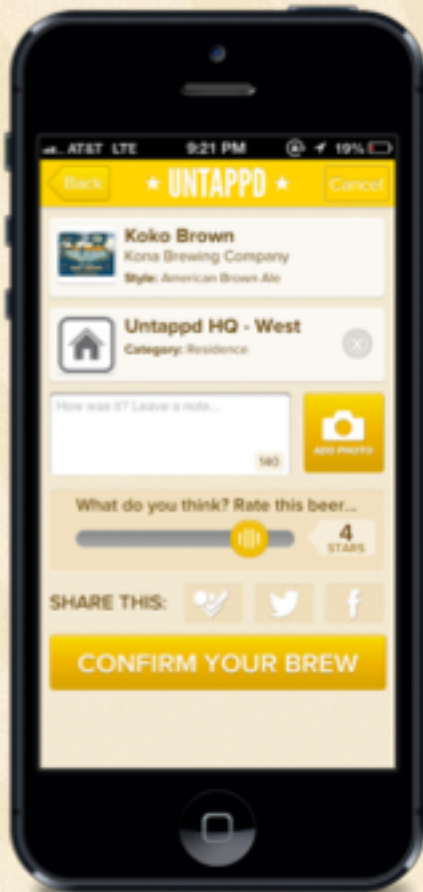
What is Untappd?

A social discovery and sharing network for beer drinkers



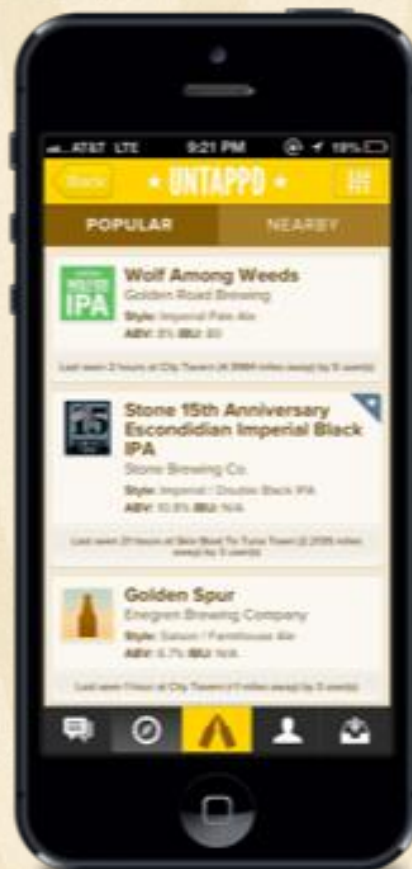
What is Untappd?

Users “check-in” to their beers, add their location, a photo, rating, comment and then share it with their friends



What is Untappd?

Using GPS, Untappd will find local and popular bars, beers and breweries nearby, where ever you are!



What are we going to talk about?

MongoDB and MySQL together

Improving on MySQL with MongoDB

- Faster joins for friends feeds
- Solving contention issues during high traffic times
- Location data, first class citizen in MongoDB
- Data Modeling differences

Scaling with ObjectRocket

MySQL and MongoDB together

It's not one or the other

- What works best for the workflow?
 - MySQL worked best for reference data for us
 - Not everything moved to MongoDB

What stayed in MySQL?

Check-ins
Users
Relationships Data
Primary Datastore

What moved to MongoDB?

Activity Feed (Friend's Graph)
Recommendation Data
Location-based Check-ins

Faster joins in Friends Feed

Background: Relational queries
(2-5 seconds per query)

- Worked well when we were small
- Friends became important as we grew
- Growth in user base affected query performance
- Friends feed
 - First thing people see in app
 - Speed is important
 - Two rows for every friendship
 - You don't want to wait Saturday night at the bar!

Today: MongoDB queries
(0.3-0.5 seconds per query)

- Friends become important
- “Schema-less” design
- One query to one collection with a shard key
- More consumable by others who develop on top of API
 - JSON

**Single query, not
multiple joins**

Contention issues in MySQL

Built for the Weekend

- Our activity is compressed to weekends and nights
 - Always looking for the next bottleneck
 - Keep looking for ways to streamline our process
 - Use MongoDB and MySQL for best performance
 - Improving Efficiency with Paging

Need for Speed and Performance

- Two key scenarios
 - Loading of friends feed
 - Checking on a beer
- Initially found lock contention
 - Moved data to MongoDB

We were averaging around 7-8k queries per second per box without MongoDB. After migration, we dropped to around 2-3k queries per box.

Location data in MongoDB

Query: “Who is nearby?”

- Location data in MongoDB
 - Anything with a GPS location is moved to MongoDB (outside of friends)
 - Queries such as “Who is nearby?” become easy
 - Gets really complicated with beer
 - LAT/LONG attached
 - It is not like a location

Query: “What beers are available close to me?”

- Geospatial index
 - What beers are popular in my area?
 - What bars are trending in my area?
 - Unique users checking into a location
- Queries look like English
 - NEAR
 - SQL queries can go bad fast
 - Location is a first class citizen in MongoDB

Data Modeling

Modeling with MongoDB

- Struggled initially storing beers
- Handling two data sources with volatile data can be difficult
- Use of TTL
 - Documents expiration
 - You are not looking at what your friends were drinking weeks ago

Handling Updates

- Identifying which fields can MySQL to back-fill
- Deletion Scripts with Message Queues to increase performance
- TTL helps with older records

Scaling with MongoDB and ObjectRocket

- History
 - Started with 5GB plan
 - Two: 100GB and 20GB
 - 100GB holds Activity Feed Data, while 20GB holds recommendations and location data
- Huge performance gains moving to the ObjectRocket architecture

In Summary


- Use MySQL and MongoDB for the right jobs
- Look for opportunities to improve query speed
- Think of complicated MySQL features that are easy in MongoDB
 - Dynamic schema
 - Location data
 - Queries
- Scalability

Thank you!



greg@untappd.com

untappd.com/user/gregavola

 @gregavola

Q&A

What should you do next?

<http://www.rackspace.com/mongodb>

I want to watch a demo of ObjectRocket

- <http://goo.gl/oxpsbA>

I am interested in migrating to ObjectRocket

- Send an email to support@objectrocket.com

I want to learn more about ObjectRocket

- Twitter [@objectrocket](https://twitter.com/objectrocket)
- Send an email to support@objectrocket.com

I want to learn more about MongoDB

- Visit www.mongodb.com for great learning and training resources

THANK YOU



RACKSPACE® HOSTING | 5000 WALZEM ROAD | SAN ANTONIO, TX 78218
US SALES: 1-800-961-2888 | **US SUPPORT:** 1-800-961-4454 | **WWW.RACKSPACE.COM**