THE MYSQL ECOSYSTEM AT GITHUB
LEAD ENGINEER @ GITHUB

- [github.com/samlambert](https://github.com/samlambert)
- [samlambert.com](http://samlambert.com)
- [twitter.com/isamlambert](https://twitter.com/isamlambert)
WHAT IS GITHUB?
> Code Hosting
> Collaboration
> Octocats
> 6+ MILLION USERS
> 15.7 MILLION REPOSITORIES
> 100+ TB OF GIT DATA
> 239 GITHUBBERS
> 100 ENGINEERS
GITHUB.COM/MYSQL/MYSQL-SERVER
THE TEAM
> SMALL TEAM ~ 15 PEOPLE
> RESPONSIBLE FOR SCALING, AUTOMATION, PAGER ROTATION, GIT STORAGE AND SITE RELIABILITY
> SUB TEAM: THE DATABASE INFRASTRUCTURE TEAM
> SHOUT OUT TO @DBUSSINK
THE GITHUB STACK
THE STACK

- Git (obviously)
- Ruby/Rails for GitHub.com
- C spread around the stack
- Puppet for provisioning
- Bash and Ruby for scripting
- Elasticsearch for .com search
- Haystack for exceptions
- Resque for queues
> RUBY ON RAILS

> GITHUB/GITHUB
> 203 CONTRIBUTORS
> 192,000 COMMITS
> LARGE RAILS APP
> ACTIVE RECORD
OBJECT RELATIONAL MAPPER

AVOCIDS WRITING SQL DIRECTLY

CAN WRITE SOME TERRIBLE QUERIES

SINGLE DB HOST APPROACH
FAST CHANGING CODEBASE
HUNDREDS OF DEPLOYMENTS A DAY
TOOLING IS EXTREMELY IMPORTANT
SELECT DATE_SUB(NOW(), INTERVAL 18 MONTH);
MAJORITY OF QUERIES SERVED FROM ONE HOST

REPLICAS USED FOR BACKUPS/FAILOVER

OLD HARDWARE/DATACENTER
> UNSCALABLE
> CONTENTION PROBLEMS
> TRAFFIC BURSTS CAUSED QUERY RESPONSE TIMES TO GO UP
TIME FOR CHANGE
needed to move data centers

chance to update hardware

new start = a chance to tune

time to functionally shard
SHARDING?

- A large volume of writes came from a single events table
- Constantly growing
- No joins
> REPLICATE TABLE DO
> MOVE READS ONTO NEW CLUSTER
> THEN FINALLY CUT WRITES OVER
> STOP REPLICATION
NOW THERE WERE TWO

> MULTIPLE CLUSTERS SHARDED FUNCTIONALLY
> SEPARATE CONCERNS
> SCALE WRITES AND READS
> EVENTS OUT OF THE WAY TIME FOR THE BIG SHOW
> THE MAIN CLUSTER WAS NEXT
BARE METAL

- NEW HARDWARE
- SSDS
- 💩 LOADS OF RAM
- 10GB NETWORKING
BUILD THE TOPOLOGY

> SINGLE MASTER
> LOTS OF READ REPLICAS
> DELAYED REPLICAS
> LOGICAL BACKUP HOSTS
> FULL BACKUP HOSTS
> REgression testing is essential
> replay queries from live cluster
> long benchmarks: 4 hours +
> one change at a time
GO LIVE

> MAINTENANCE WINDOW
> 13 MINUTES
TIME TO USE THAT HARDWARE
START

[Diagram showing a network with START, MASTER, and APPS nodes connected by arrows indicating communication and replication.]
NEW DESIGN
How do you transition a monolithic app to use multiple database hosts?
CONNECTIONS

> Split out the current connection
> Write
> Read only
> We made the decision to have all GET requests use a replica.
> All posts and gets after a post for a user use the master.
> After 3 seconds the user moves to a replica.
WE WANTED TO TAKE THE SMALLEST STEPS POSSIBLE EACH TIME
WE VERIFIED OUR CHANGES AT EACH STEP IN THE PROCESS
> HOW DO WE KNOW WE AREN’T GOING TO BREAK ANYTHING?
> WE SET UP A CONNECTION WE CALLED “WRITE ALERT”
> WRITE ALERT ALLOWED WRITES BUT NOTIFIED US
Haystack is our exception tracking tool backed by Elasticsearch and awesome.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>class_name</td>
<td>QueryWriteAlertLogger::QueryWriteAlertError</td>
</tr>
<tr>
<td>controller</td>
<td>Api::Users</td>
</tr>
<tr>
<td>current_ref</td>
<td>dbussink/api_database_selection_middleware</td>
</tr>
<tr>
<td>language</td>
<td>nil</td>
</tr>
<tr>
<td>method</td>
<td>GET</td>
</tr>
<tr>
<td>params</td>
<td>{}</td>
</tr>
<tr>
<td>query</td>
<td>INSERT INTO <code>table</code> ... statements</td>
</tr>
<tr>
<td>referrer</td>
<td>nil</td>
</tr>
<tr>
<td>remote_ip</td>
<td></td>
</tr>
<tr>
<td>repo</td>
<td>false</td>
</tr>
<tr>
<td>request_id</td>
<td></td>
</tr>
</tbody>
</table>
WRITE ALERTS

▶ THIS ALLOWED US TO TEST MOVING TO A READ ONLY CONNECTION WITHOUT IMPACTING USERS
▶ WE FIXED ANY ISSUES THAT CAME UP
▶ WHEN WE STOPPED GETTING ALERTS WE KNEW WE WERE READY TO GO READ ONLY
WE STAFF SHIP FEATURES AND CHANGES TO HELP US GAIN CONFIDENCE
haproxy

needed a way of distributing queries among replicas

plenty of prior art
We created HAProxy pairs for HA and failover.
>
WE STARTED WITH A SUBSET OF OUR APP
>
A PROXY THAT CHECKS YOU HAVE PERMISSIONS TO PUSH AND PULL TO A REPO
>
READ INTENSIVE
> SLOW RAMP UP
> > 1%
> > 5%
Heartbeat

- Permissions are replication sensitive
- pt-heartbeat
- Gitaauth checks
- 1 second of delay = move back to the master
BUILD CONFIDENCE

▷ REST OF THE APP HAD TO FOLLOW
▷ KEEP UPPING THE %
FAILOVER
PSUs

> PARTS GO 💥
> MORE PARTS TO KEEP GITHUB UP
- Pause the request
- Reconnect through the proxy
PERFORMANCE DEGRADATION
keeping an eye

> graphing at github is awesome
> shout out to @jssjr github.com/jssjr
WE NOTICED AN UPWARD TREND IN LATENCY
median @babeld.gitauth.curl-time (milliseconds)

- curl-time
- wait-time
 › Hasn’t always worked well in the past
 › Connections tended to stick to a process
> UPGRADES WERE REQUIRED FOR BETTER BALANCE
SLOW AND STEADY

- Deploy app to use upgraded secondary haproxy
- Roll through the cluster
THE DOWN SIDES
HURRY UP

> REPPLICATION DELAY IS PAINFUL
> BE CAREFUL WHERE YOU CAN TOLERATE DELAY
> LARGE UPDATES, INSERTS, DELETES
> DEPENDENT DESTROY
> TRANSITIONS
> DELAY IS PAINFUL
> BE CAREFUL WHERE YOU CAN TOLERATE DELAY
Remedy

> GET AFTER A POST GETS A MASTER
We modified the app when a statement modifies too many rows we send it to Haystack.

Insight
<table>
<thead>
<tr>
<th>action</th>
<th>NotificationsController#mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td>text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,&quot;/*;q=0.8</td>
</tr>
<tr>
<td>areas_of_responsibility</td>
<td>[redacted]</td>
</tr>
<tr>
<td>current_ref</td>
<td>master</td>
</tr>
<tr>
<td>dbconn</td>
<td>write</td>
</tr>
<tr>
<td>method</td>
<td>POST</td>
</tr>
<tr>
<td>oauth_access_id</td>
<td>nil</td>
</tr>
<tr>
<td>params</td>
<td>[redacted]</td>
</tr>
</tbody>
</table>
developers need to modify data
must be replication safe
query haproxy
check replicas
Contributions

- Email change
- Active users caused delay
- Support request
- Use the throttler
KEEPING THINGS FAST
Tooling is essential. Never underestimate the power of being able to write tools.
Log it

> We built a slow query logger into the app
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>accept</code></td>
<td>application/vnd.github.v3+json, application/vnd.github.beta+json, q=0.5, application/json, q=0.1</td>
</tr>
<tr>
<td><code>areas_of_responsibility</code></td>
<td>[api, :commit_statuses]</td>
</tr>
<tr>
<td><code>current_ref</code></td>
<td>master</td>
</tr>
<tr>
<td><code>database_connection</code></td>
<td>github_production</td>
</tr>
<tr>
<td><code>method</code></td>
<td>GET</td>
</tr>
<tr>
<td><code>params</code></td>
<td>[ ]</td>
</tr>
<tr>
<td><code>rails</code></td>
<td>3.0.20.github12</td>
</tr>
<tr>
<td><code>referrer</code></td>
<td>nil</td>
</tr>
<tr>
<td><code>remote_ip</code></td>
<td>146.146.146.146</td>
</tr>
<tr>
<td><code>repo</code></td>
<td>[ ]</td>
</tr>
<tr>
<td><code>request_id</code></td>
<td>nil</td>
</tr>
<tr>
<td><code>route</code></td>
<td>[ ]</td>
</tr>
<tr>
<td><code>ruby</code></td>
<td>ruby 2.1.2p95-rc1 (71c08a81c0) [x86_64-linux]</td>
</tr>
</tbody>
</table>
> Developer on call
> A spike in needles pages someone
STAFF MODE
SEE ALL QUERIES ON A PAGE
WITH TIMES
GITHUB.COM/PEEK/PEEK
<table>
<thead>
<tr>
<th>START</th>
<th>DURATION</th>
<th>RESULTS</th>
<th>QUERY</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.400ms</td>
<td>0.329ms</td>
<td>1</td>
<td><code>SELECT * FROM `repos` WHERE `repo_id` = 123456;</code></td>
</tr>
<tr>
<td>7.168ms</td>
<td>0.359ms</td>
<td>1</td>
<td><code>SELECT * FROM `repos` WHERE `repo_id` = 123457;</code></td>
</tr>
<tr>
<td>8.206ms</td>
<td>0.349ms</td>
<td>1</td>
<td><code>SELECT * FROM `repos` WHERE `repo_id` = 123458;</code></td>
</tr>
<tr>
<td>9.405ms</td>
<td>0.410ms</td>
<td>1</td>
<td><code>SELECT * FROM `repos` WHERE `repo_id` = 123459;</code></td>
</tr>
<tr>
<td>12.262ms</td>
<td>0.412ms</td>
<td>1</td>
<td><code>SELECT * FROM `repos` WHERE `repo_id` = 123460;</code></td>
</tr>
</tbody>
</table>
SLOW TRANSACTIONS
- Query Pile Up
- Site Stalls
- Bad User Experience
WE NOTICED TWO ISSUES:
- TABLE STATS
- METADATA LOCKING
> INNODB_STATS_ON_METADATA
> INNODB_STATS_AUTO_UPDATE
> GITHUB.COM/SAMLAMBERT/PT-ONLINE-SCHEMA-CHANGE-ANALYZE
> QUERIES PILED UP BEHIND A METADATA LOCK
< TABLE COPY AND SWAP
mysql> show tables;
+---------------------+
| Tables_in_test      |
+---------------------+
| test                |
| test_new            |
+---------------------+
2 rows in set (0.00 sec)
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM test;
Empty set (0.00 sec)
mysql> RENAME TABLE test TO test_old, test_new TO test;
Query OK, 0 rows affected (2 min 48.57 sec)
<table>
<thead>
<tr>
<th>Id</th>
<th>User</th>
<th>Host</th>
<th>db</th>
<th>Command</th>
<th>Time</th>
<th>State</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>samlambert</td>
<td>localhost:56703</td>
<td>test</td>
<td>Sleep</td>
<td>195</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>19</td>
<td>samlambert</td>
<td>localhost:56715</td>
<td>test</td>
<td>Query</td>
<td>33</td>
<td>Waiting for table metadata lock</td>
<td>RENAME TABLE test TO test_old, test_new TO test</td>
</tr>
<tr>
<td>21</td>
<td>root</td>
<td>localhost:56727</td>
<td>test</td>
<td>Sleep</td>
<td>23</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>22</td>
<td>root</td>
<td>localhost:56728</td>
<td>NULL</td>
<td>Sleep</td>
<td>23</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>24</td>
<td>root</td>
<td>localhost:56750</td>
<td>NULL</td>
<td>Query</td>
<td>0</td>
<td>NULL</td>
<td>show processlist</td>
</tr>
</tbody>
</table>

5 rows in set (0.01 sec)
<table>
<thead>
<tr>
<th>Id</th>
<th>User</th>
<th>Host</th>
<th>db</th>
<th>Command</th>
<th>Time</th>
<th>State</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>samlambert</td>
<td>localhost:56703</td>
<td>test</td>
<td>Sleep</td>
<td>233</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>19</td>
<td>samlambert</td>
<td>localhost:56715</td>
<td>test</td>
<td>Query</td>
<td>71</td>
<td>Waiting for table metadata lock</td>
<td>RENAME TABLE test TO test_old, test_new TO test</td>
</tr>
<tr>
<td>21</td>
<td>root</td>
<td>localhost:56727</td>
<td>test</td>
<td>Sleep</td>
<td>2</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>22</td>
<td>root</td>
<td>localhost:56728</td>
<td>test</td>
<td>Sleep</td>
<td>61</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>24</td>
<td>root</td>
<td>localhost:56750</td>
<td>NULL</td>
<td>Query</td>
<td>0</td>
<td>NULL</td>
<td>show processlist</td>
</tr>
<tr>
<td>25</td>
<td>samlambert</td>
<td>localhost:56761</td>
<td>test</td>
<td>Query</td>
<td>4</td>
<td>Waiting for table metadata lock</td>
<td>SELECT * FROM test</td>
</tr>
</tbody>
</table>

6 rows in set (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> RENAME TABLE test TO test_old, test_new TO test;
Query OK, 0 rows affected (2 min 48.57 sec)

mysql> SELECT * FROM test;
Empty set (1 min 41.70 sec)
PREVENTION

- SMALLER TRANSACTIONS
- DETECTION
at app/api/statuses.rb:90 (took 448ms = 446ms + 1ms idle)

accept: application/vnd.github.v3+json, application/vnd.github.beta+json; q=0.5, application/json; q=0.1

areas_of_responsibility: [api, :commit_statuses]
current_ref: master

method: POST

params: 

queries:  

  0.115 | BEGIN | 0.107ms  
  442.322 | UPDATE | 0.232ms  
  443.116 | INSERT INTO | 4.650ms

rails: 3.0.20, github12

referrer: nil

remote_ip: 

repo: 

request_id: 

CHATOPS
MEET HUBOT

> NODE.JS
> OPEN SOURCE
> GITHUB.COM/GITHUB/HUBOT
> HUNDREDS OF PLUGINS
I have found the things
SHOW AND TELL

- IT ALL HAPPENS IN CHAT
- AMAZING FOR LEARNING
- SHARE THE TERMINAL
ANYTHING

- DROP TABLES
- SEE WHO’S IN THE OFFICE
- DEPLOY APPS
> Chat is central to our culture
> 52% of GitHub is remote
> How do you give everyone context?
> Safe
> Intuitive
> Accessable
> People will use it
EXPLAIN QUERIES VIA HUBOT
<table>
<thead>
<tr>
<th>Select Type</th>
<th>Table</th>
<th>Type</th>
<th>Possibl...</th>
<th>Key</th>
<th>Key Len</th>
<th>Ref</th>
<th>Rows</th>
<th>Ext...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td>users</td>
<td>ALL</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>...</td>
</tr>
</tbody>
</table>
EXPLAIN

> LEARN TOGETHER
> WORK AS A TEAM
> NO NEED FOR A MEETING/EMAIL
PROFILE

PROFILE QUERIES
### Hubot

**samlambert**

```sql
/profile! SELECT * FROM users LIMIT 1
```

<table>
<thead>
<tr>
<th>Status</th>
<th>Duration (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>starting</td>
<td>0.000012</td>
</tr>
<tr>
<td>checking permissions</td>
<td>0.000003</td>
</tr>
<tr>
<td>Opening tables</td>
<td>0.000022</td>
</tr>
<tr>
<td>System lock</td>
<td>0.000012</td>
</tr>
<tr>
<td>init</td>
<td>0.000017</td>
</tr>
<tr>
<td>optimizing</td>
<td>0.000001</td>
</tr>
</tbody>
</table>

[20 more lines]
GITHUB.COM/SMALAMBERT/HUBOT-MYSQL-CHATOPS
You do not have to write CoffeeScript!
> 34279 lines of Ruby and Shell
> Wrapped by Hubot
TRUNCATE

> SAFE
> VISIBLE
> REPEATABLE
"CREATE TABLE _new_users LIKE users"
"ALTER TABLE _new_users AUTO_INCREMENT = 1"
"RENAME TABLE users TO _users_old<date>, _new_user TO users"
/mysql backup users

Hubot

A backup job has been created for the users table. You will be notified in the Ops room when the backup is complete.
<table>
<thead>
<tr>
<th>Backup ID</th>
<th>Table Name</th>
<th>Donor Host</th>
<th>Backup Start</th>
<th>Backup End</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>178602</td>
<td>users</td>
<td></td>
<td>2014-03-29 08:23:36</td>
<td>2014-03-29 08:29:13</td>
<td></td>
</tr>
<tr>
<td>178263</td>
<td>users</td>
<td></td>
<td>2014-03-29 00:22:18</td>
<td>2014-03-29 00:26:46</td>
<td></td>
</tr>
<tr>
<td>177251</td>
<td>users</td>
<td></td>
<td>2014-03-28 00:24:10</td>
<td>2014-03-28 00:28:54</td>
<td></td>
</tr>
</tbody>
</table>
[324 more lines]
backup

＞ NO EXCUSE
＞ AVAILABLE TO ANYONE
＞ USES AN APP CALLED SAFEHOLD
SAFEHOLD

> FIRES BACKUP JOBS INTO A QUEUE
> WORKERS WORK ON DIFFERENT TYPES OF JOBS
RESTORE

> RESTORE ANY LOGICAL BACKUP
> BACKUPS GO TO INTERMEDIATE HOSTS
samlambert
/mysql restore 178602

Hubot
A restore job has been created for the backup job 178602. You will be notified in the Ops room when the restore is complete.
› CLONE TABLES ONTO TEST SERVERS
› GREAT FOR TESTING INDEXES
› DEVELOPERS USE THIS A LOT
Proxy Control

- Weight servers
- Take them from the pool
/DEPLOY
/GRAPH ME -1H @MYSQL.RWPS
technoweenie
daarn nice debugging

rtomayko
really need to gut a ton of it and figure out how to do things sanely

jnnunemaker
yeah
no small task

/graph me -1w aliasSub(substr(limit(sortBy(Maxima(sumSeriesWithWildcards((hosts,collectd).github_*.mysql- github.mysql_commands~(update,commit,begin).value,1)),8),2,3),"mysql_commands~"," ") + title="%40mysql.commands(github_.github..)"&areaMode=stacked&areaAlpha=0.5&yMin=0&hideLegend=false&template=plain

Hubot

@mysql.commands(github_.github..)

samlambert
rtomayko: '+'

jnunemaker
i'm stoked that my new dashboard actually helped me
haha
/STATUS YELLOW <MESSAGE>
> LETTING YOU ALL KNOW
MITIGATE

> ATTACKS HAPPEN
> WHY GET SAD?
> USE THE CHATOPS
QUESTIONS?