

The Post-Incident Review

Issue 3: April 2020

The Post-Incident Review

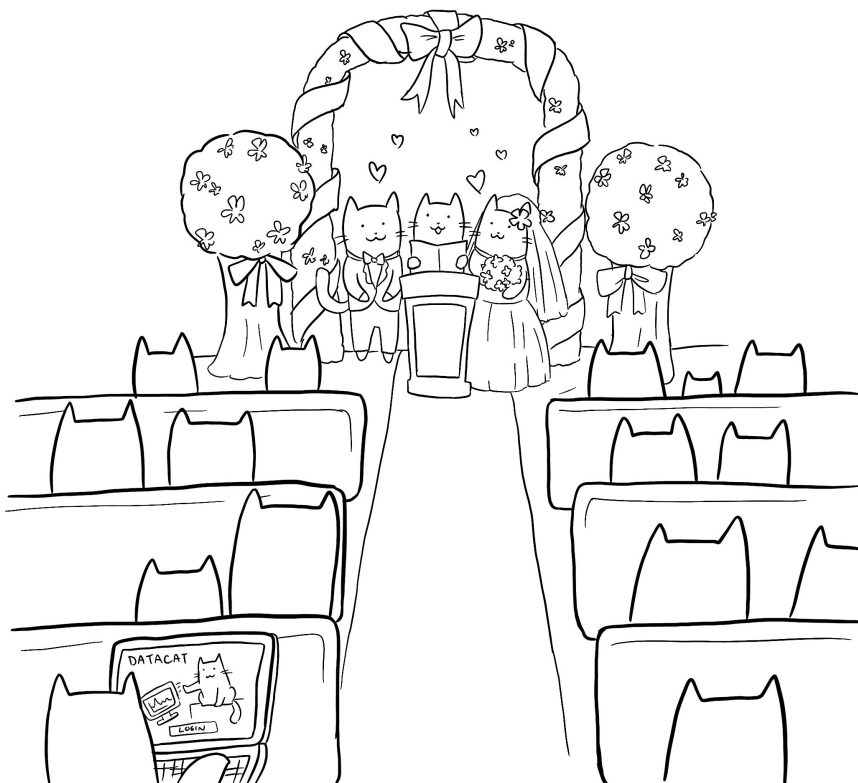
Issue 3: April 2020

- Page 2.** Featured Illustration:
“Humans of On-Call”
- Page 3.** GitHub: “February service
disruptions post-incident
analysis”
- Page 10.** Feature Article: “What Does
Fairness Mean for On-call
Rotations?”
- Page 12.** The Incident Labs Bookshelf

We're Incident Labs, and we're interested in figuring out the best practices for incident management for software companies. We're building Ovvv Insights, which provides on-call intelligence for healthier teams..

Got comments, suggestions, or compliments?
Drop us a line at zine@incidentlabs.io!

HUMANS
OF
ON-CALL



Shout out to Andrew Louis (@saucinonyouuuu), who inspired this illustration with his story of getting paged while in a church!
Color in this illustration by Denise Yu (@DeniseYu21)!

“February service disruptions post-incident analysis”

Keith Ballinger, GitHub

Published on March 26, 2020

In late February, GitHub experienced multiple [service interruptions](#) that resulted in degraded service for a total of eight hours and 14 minutes over four distinct events. Unexpected variations in database load, coupled with an unintended configuration issue introduced as a part of ongoing scaling improvements, led to resource contention in our mysql database cluster.

Background

Originally, [all our MySQL data lived in a single database cluster](#). Over time, as mysql grew larger and busier, we split functionally grouped sets of tables into new clusters and created new clusters for new features. However, much of our core dataset still resides within that original cluster.

[We're constantly scaling our databases to handle the additional load](#) driven by new users and new products. In this case, an unexpected variance in database load contributed to cluster degradations and unavailability.

Timeline (all times are UTC)

2020 February 19 15:17 (lasting for 52 minutes)

At this time, an unexpectedly resource-intensive query began running against our mysql database cluster. The intent was to run this load against our read replica pool at a much lower frequency, but we inadvertently sent this traffic to the master of the cluster, increasing the

GitHub: “February service disruptions
post-incident analysis”

pressure on that host beyond surplus capacity. This pressure overloaded ProxySQL, which is responsible for connection pooling, resulting in an inability to consistently perform queries.

2020 February 20 21:31 UTC (lasting for 47 minutes)

Two days later, as part of a planned master database promotion, we saw unexpectedly high load which triggered a similar ProxySQL failure. The intent of this maintenance was to give our teams visibility into issues they might experience when a master is momentarily read-only.

After an initial load spike, we were able to use the same remediation steps as in the previous incident to restore the system to a working state. We suspended further maintenance events of this type as we investigated how this system failed.

2020 February 25 16:36 UTC (lasting for 132 minutes)

In this 3rd incident involving ProxySQL, active database connections crossed a critical threshold that changed the behavior of this new infrastructure. Because connections remained above the critical threshold after remediation, the system fell back into a degraded state. During this time, GitHub.com services were affected by stalled writes on our mysql database cluster.

As a result of our previous investigation, we understood that file descriptor limits on ProxySQL nodes were capped at levels significantly lower than intended and insufficient to maintain throughput at high load levels. Specifically, because of a system-level limit of 1048576, our process manager silently reduced our LimitNOFILE

setting from 1073741824 to 65536. During remediation, we also encountered a race condition between our process manager and service configurations which slowed our ability to change our file limit to 1048576.

2020 February 27 14:31 UTC (lasting for 263 minutes)
Application logic changes to database query patterns rapidly increased load on the master of our mysql database cluster. This spike slowed down the cluster enough to affect availability for all dependent services.

What we learned

Observability and performance improvements

We've used these events to identify operational readiness and observability improvements around how we need to operate ProxySQL. We have made changes to allow us to more quickly detect and address issues like this in the future. Remediating these issues were straightforward once we tracked down interactions between systems. It's clear to us that we need better system integration and performance testing at realistic load levels in some areas before fully deploying to production.

We're also devoting more energy to understanding the performance characteristics of ProxySQL at scale and the trickle-down effect on other services before it affects our users.

Optimizing for stability

We decided to freeze production deployments for three days to address short-term hotspotting as a result of the final incident on February 27. This helped us stabilize GitHub.com. Our increased confidence in service

reliability gave us the breathing room to plan next steps thoughtfully and also helped identify long-term investments we can make to mitigate underlying scalability issues.

Next Steps

Immediate changes: Data partitioning

We shipped a sizable chunk of data partitioning efforts we’ve worked on for the past six months just days after these incidents for one of our more significant MySQL table domains, the “abilities” table. Given that every authentication request to GitHub uses this table domain in some way, we had to be very deliberate about how we performed this work to fulfill our requirement of zero downtime and minimal user-impact.

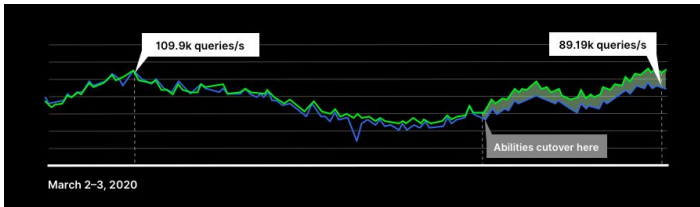
To accomplish this, we:

1. Removed all JOIN queries between the `abilities` table and any other table in the database
2. Built a new cluster to move all of the table data into
3. Copied all data to the new cluster using Vitess’s vertical replication feature, keeping the copied data up-to-date in real-time
4. Moved all reads to the new cluster
5. Moved all writes to the new cluster using Vitess’ proxy layer `vtgate`

Steps 1 and 2 took months’ worth of effort, while steps 3 through 5 were completed in four hours.

These changes reduced load on the `mysql1` cluster master by 20 percent, and queries per second by 15 percent. The following graph shows a snapshot of query

performance on March 2nd, prior to partitioning, and on March 3rd, after partitioning. Queries per second peaked at 109.9k/second immediately prior to partitioning and decreased to a peak of 89.19k queries/second after.



Additional technical and organizational initiatives

- **Audit and lower reads from leader databases**

We're auditing reads from master databases with the intent to lower them. We put unnecessary load on our master databases by reading from them when replicas have the same data. This can exacerbate capacity issues and service reliability by adding load to the most critical databases at GitHub. By auditing and changing these reads to replica databases, we increase headroom in our SQL databases and reduce the risk of them being overloaded as production load varies.

- **Widen our usage of feature flags**

We're requiring feature flags for code updates, which allows us to disable problematic code dynamically, and will dramatically speed up recovery for active incidents.

Additional technical and organizational initiatives (continued)

- **Complete in-flight functional partitioning**

We’re completing our in-flight functional partitioning of our mysql database cluster. We’ve been working on moving tables that comprise functional domains out of the cluster over the last year. We’re very close to finishing that work for a significant number of tables and schema domains. Shipping these improvements will immediately reduce writes to the cluster by 60 percent and storage requirements by 70 percent. Expect to see another post in the coming months that details the performance benefits from this work.

- **Refine our dashboards**

We’re improving our visibility into the effects of deploys on our mysql database cluster. Our current deployment dashboard can be noisy, making it difficult to determine deploy safety. Interviews with engineers involved in these incidents noted the cognitive load required to process noisy dashboards. We predict refined dashboards will allow us to spot problems with deploys earlier in the process.

- **Invest in additional data partitioning opportunities**

We’ve identified an additional 12 schema domains that we’ll split out from the cluster to further protect us from request and storage constraints.

Additional technical and organizational initiatives (continued)

- **Shard to scale**

We'll begin sharding our largest schema set. Functional partitioning buys us time, but it's not a sufficient solution to our scaling needs. We plan to use sharding (tenant partitioning) to move us from vertical scaling to horizontal scaling, enabling us to expand capacity much more easily as we grow.

In summary

We know you depend on our platform to be reliable. With the immediate changes we've made and long-term plans in progress, we'll continue to use what we've learned to make GitHub better every day.

Originally printed at:

<https://github.blog/2020-03-26-february-service-disruptions-post-incident-analysis/>

What Does Fairness Mean for On-call Rotations?

By Jaime Woo

Over the winter, Emil and I took a road trip for a friend's housewarming. The drive was five hours—not bad with two of us. My leg there was easy. Emil's leg back, not so much. Three hours in, we hit a snowstorm. Driving is stressful and exhausting during a blizzard. When we saw a sign for an upcoming rest stop, I asked Emil if he wanted me to take over.

Depending on the task, there are many ways to divide labour. For a road trip, we expected to roughly spend equal amounts of time driving. If Emil accepted my offer, I would then end up with more total driving time. However, that time was mostly smooth sailing, obviously lower impact than a winter storm.

When reflecting on how to characterize burden, it dawned on us that many teams go through a similar calculus for incident response, in how to fairly assign on-call rotations.

The most common method is to base rotations on fixed periods of time, but is organizing shifts solely based on time the strongest strategy? We think factoring in the experience of on-call needs to be front-and-center.

Yes, there are reasons for basing on-call schedules on equal lengths of time. The regularity provides structure and certainty. Carrying a laptop everywhere you go is inconvenient, and having a set schedule means you can

know when to book an appointment, or go on vacation.

But the need for certainty extends beyond knowing if you're on-call or not. We don't know when interruptions will happen. The random nature means that you can't guarantee an even distribution of load across a team.

When schedules are inflexible the random nature of interruptions means some people can end up taking the brunt. A schedule that equates fairness with time on shift can't take that into account. A hybrid approach that acknowledges the number of alerts and interruptions as well as time spent on-call would allow schedules to be fairer, and lead to healthier, more sustainable rotations.

This might mean a cap on incidents, such as Google does, allowing only two incidents per 12-hour shift. Or, it may make sense that after someone has faced a disproportionate burden for interruptions their next shift is rescheduled until they've had enough time to recover. Another alternative is to ensure there are always primary and secondary on-calls, and that the two people swap roles to average out the labour.

In the end, Emil was determined to complete the drive (we'll save a discussion on hero culture for another day) and we safely arrived home a few hours later. Yet even the offer to swap injected flexibility into the system. We shouldn't be afraid to ask each other how we're handling the road, and if the driver needs a break. In the end, we're all in this journey together.

Learn more about on-call intelligence at <https://ovvy.io>

The Incident Labs Bookshelf

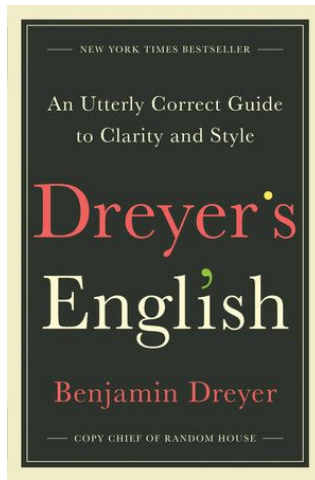
“[Dreyer’s English](#)” by Benjamin Dreyer
Random House, 2019

Why do you want to read this?

If you’re involved in post-incident reviews, you’re likely writing. Writing can be intimidating because of the many rules and stylistic quirks involved, but it needn’t be.

Ben’s book is a fantastic, enjoyable read—there are many literal laugh-out-loud moments—that will not only erase traumatic memories from English class, but will help you understand how to deploy words in a powerful, meaningful manner.

- Emil Stolarsky and Jaime Woo



**Let us know what you think of Dreyer’s English on
Twitter by tagging us @incidentlabsinc**

*A collection of public postmortems
for you to read, reflect, and annotate
wherever and whenever you'd like.*

Visit us at the Post-Incident Review website
<https://zine.incidentlabs.io>

A Project by Incident Labs