Observability

The Health of Every Request

Nathan LeClaire
nathan@honeycomb.io
twitter.com/dotpem

Overview

On Observability

Where we have come from and why does o11y matter?

o11y Report Card

How do various approaches stack up?

The Health of Every Request

Why should we care, and how do we care?

Making o11y Affordable

How do those of us with limited resources make it work?

$(whoami)

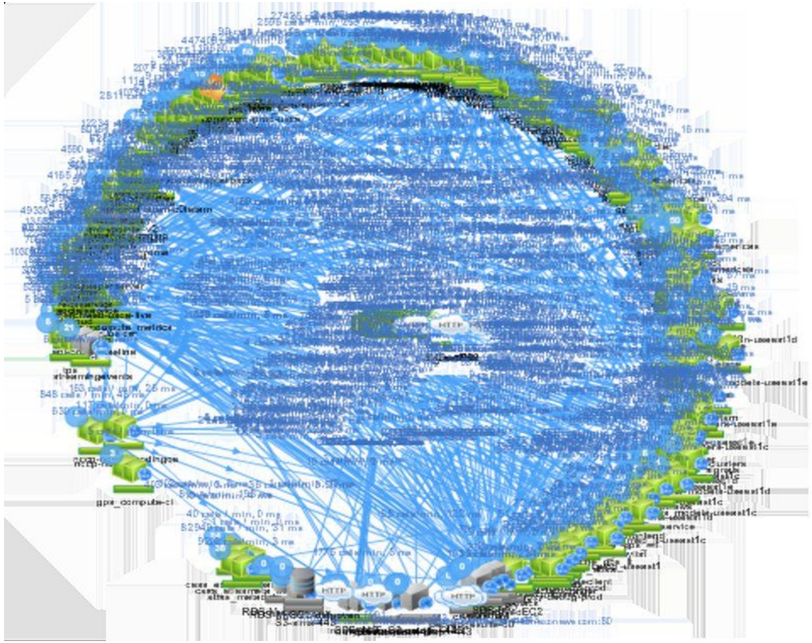
honeycomb.io
Real-time system debugging
Find and fix gnarly bugs in seconds, not days.

Nathan LeClaire

- Previously Open Source Engineer at Docker.
- Platform Engineer and Sales Engineer at Honeycomb.
- Writer of “funny” tweets @dotpem and sometimes articles at https://nathanleclaire.com.
- **Weapons of choice:** Golang, Linux debugging tools, low bar squat, “Epic & Melodic” metal playlist on Spotify.

On Observability

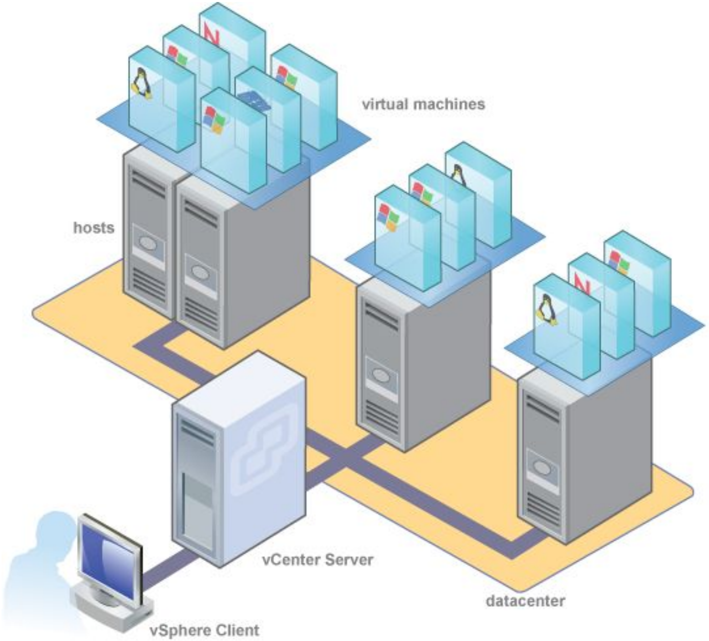
What’s the big deal with o11y?



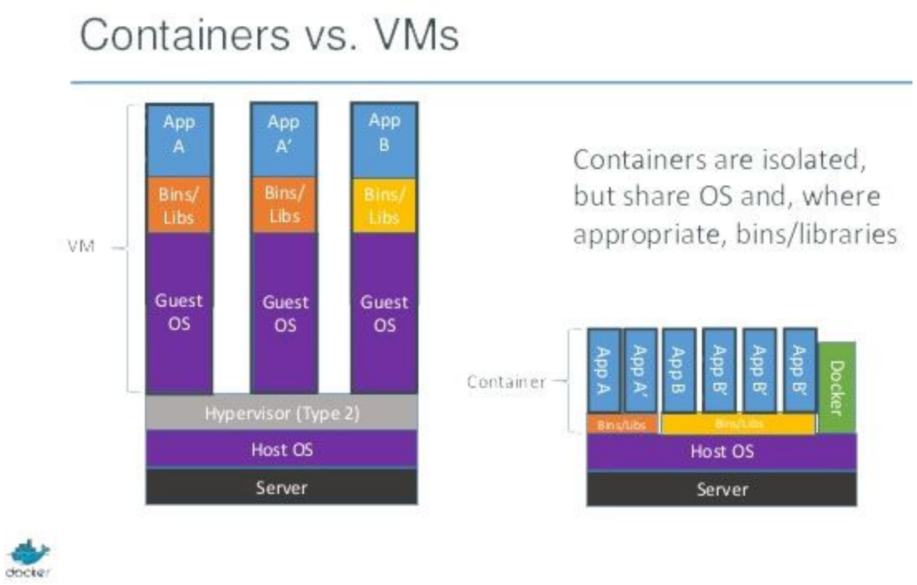
The world used to be simpler.

Debugging is so easy. I just have one server I SSH into and I use `tail` on logs. BOOM!

But then VMs happened...



… then containers happened.



Containers vs. VMs

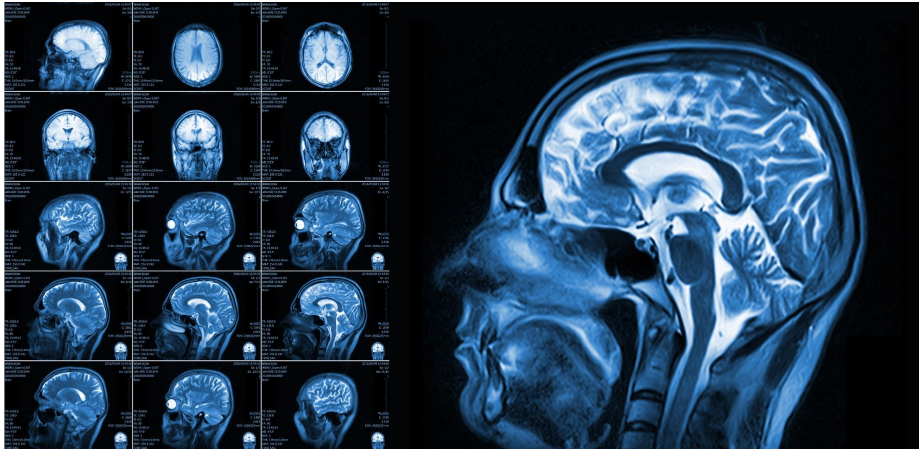
Containers are isolated, but share OS and, where appropriate, bins/libraries

Now, #Serverless is happening?

But… our o11y tools are still bad and we should feel bad.

We have monitoring but we need observability

vs.



Defining observability

Charity Majors

@mipsytipsy

"Can I ask new questions about my system from the outside, and understand what is happening on the inside - all without shipping any new code?"

More observable businesses will build better platforms

We're building an AI powered platform based on the blockch-
SHUT UP AND TAKE MY MONEY

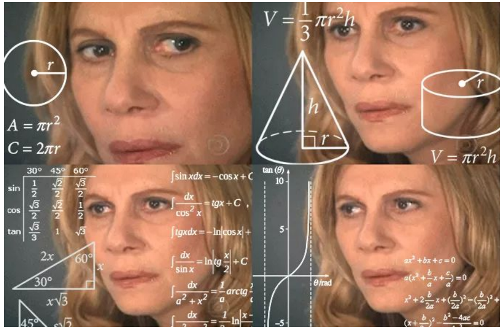
Seriously though, the winners of the future will be united by at least one common thread: they will offer more functionality and user customizability, up to and including executing arbitrary code. And more customizability comes with more o11y problems.

Just look at Shopify, or Slack, or the recently released Github Actions feature. Why would Salesforce would buy Heroku? Because they are a platform company, not a CRM company.

More observable businesses will attract better engineers

Company A:

- Devs spend most of their time writing code
- o11y gives them the confidence to deploy frequently
- o11y makes it easy to understand how your users are interacting with your code and how it's performing

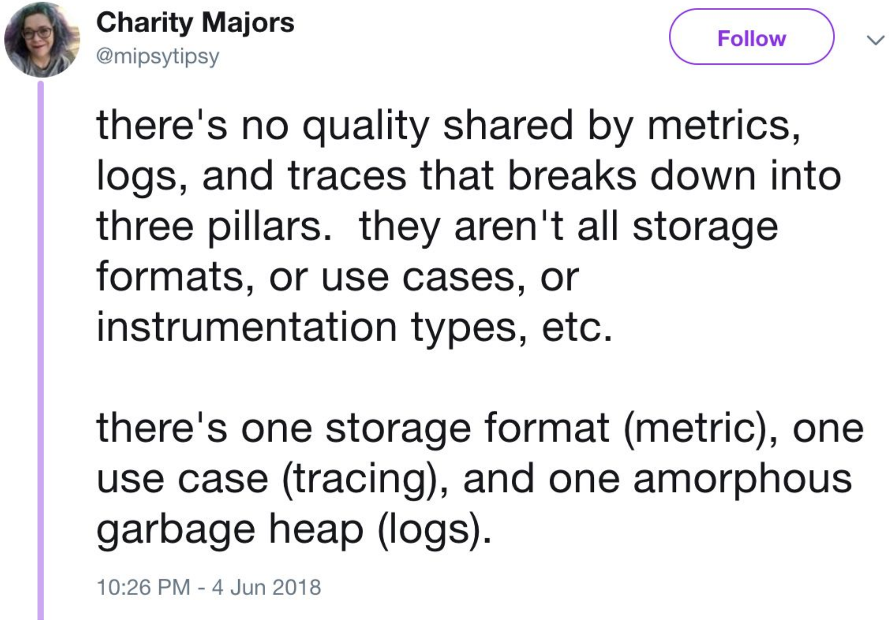


Company B:

- Devs spend most of their time firefighting
- Deploys are an infrequent occurrence because they always cause new bugs
- Engineers have very few ways to understand what their code is doing once deployed

More observable businesses will beat their competitors

“Three Pillars?”



Charity Majors
@mipsytipsy

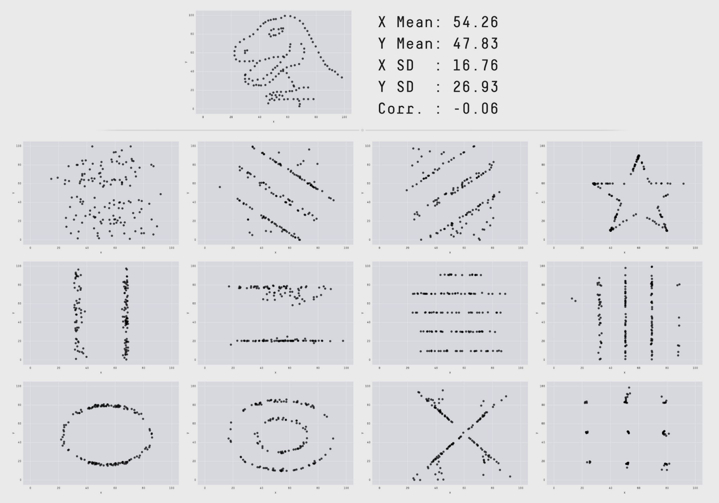
there's no quality shared by metrics, logs, and traces that breaks down into three pillars. they aren't all storage formats, or use cases, or instrumentation types, etc.

there's one storage format (metric), one use case (tracing), and one amorphous garbage heap (logs).

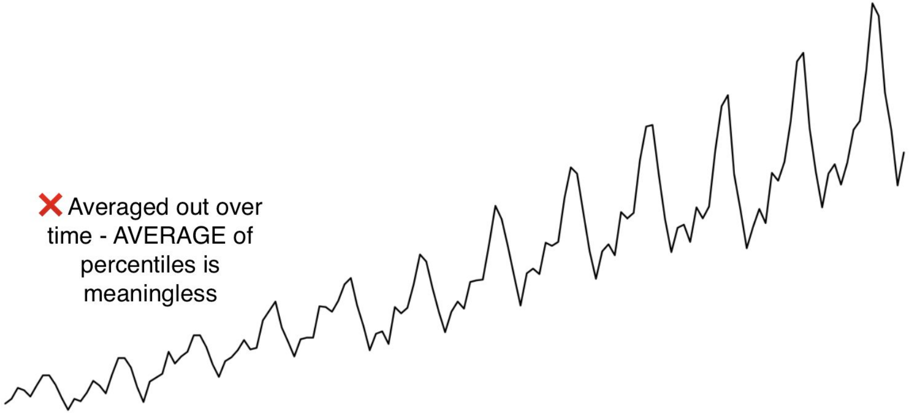
10:26 PM - 4 Jun 2018

o11y report card

Metrics - D



❌ No access to the raw data comprising the time series.

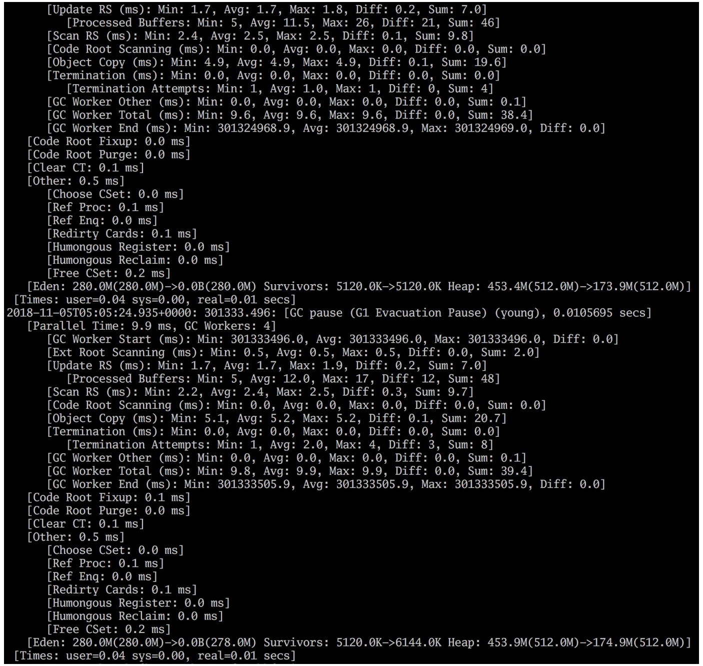


❌ Averaged out over time - AVERAGE of percentiles is meaningless

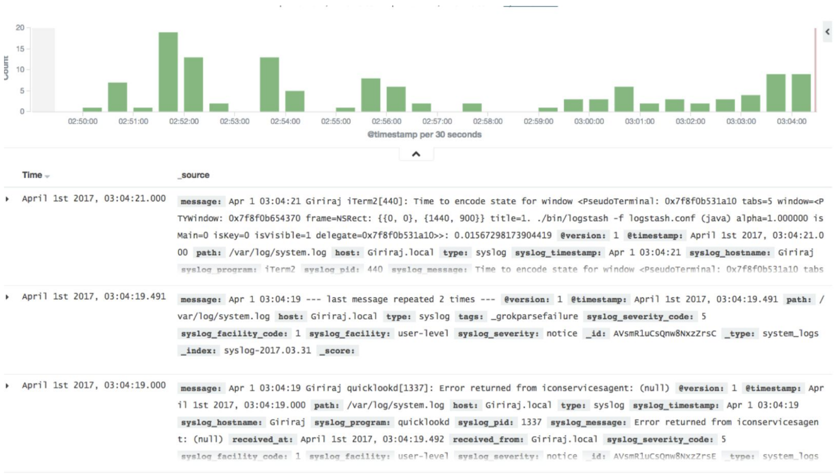
CAUTION: Remember that every unique combination of key-value label pairs represents a new time series, which can dramatically increase the amount of data stored. Do not use labels to store dimensions with high cardinality (many different label values), such as user IDs, email addresses, or other unbounded sets of values.

❌ NO HIGH CARDINALITY

Logs - C



```
      [Update RS (ms): Min: 1.7, Avg: 1.7, Max: 1.8, Diff: 0.2, Sum: 7.0]
         [Processed Buffers: Min: 5, Avg: 11.5, Max: 26, Diff: 21, Sum: 46]
      [Scan RS (ms): Min: 2.4, Avg: 2.5, Max: 2.5, Diff: 0.1, Sum: 9.8]
      [Code Root Scanning (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]
      [Object Copy (ms): Min: 4.9, Avg: 4.9, Max: 4.9, Diff: 0.1, Sum: 19.6]
      [Termination (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]
         [Termination Attempts: Min: 1, Avg: 1.0, Max: 1, Diff: 0, Sum: 4]
      [GC Worker Other (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.1]
      [GC Worker Total (ms): Min: 9.6, Avg: 9.6, Max: 9.6, Diff: 0.0, Sum: 38.4]
      [GC Worker End (ms): Min: 301324968.9, Avg: 301324968.9, Max: 301324969.0, Diff: 0.0]
   [Code Root Fixup: 0.0 ms]
   [Code Root Purge: 0.0 ms]
   [Clear CT: 0.1 ms]
   [Other: 0.5 ms]
      [Choose CSet: 0.0 ms]
      [Ref Proc: 0.1 ms]
      [Ref Enq: 0.0 ms]
      [Redirty Cards: 0.1 ms]
      [Humongous Register: 0.0 ms]
      [Humongous Reclaim: 0.0 ms]
      [Free CSet: 0.2 ms]
   [Eden: 280.0M(280.0M)->0.0B(280.0M) Survivors: 5120.0K->5120.0K Heap: 453.4M(512.0M)->173.9M(512.0M)]
 [Times: user=0.04 sys=0.00, real=0.01 secs]
2018-11-05T05:05:24.935+0000: 301333.496: [GC pause (G1 Evacuation Pause) (young), 0.0105695 secs]
   [Parallel Time: 9.9 ms, GC Workers: 4]
      [GC Worker Start (ms): Min: 301333496.0, Avg: 301333496.0, Max: 301333496.0, Diff: 0.0]
      [Ext Root Scanning (ms): Min: 0.5, Avg: 0.5, Max: 0.5, Diff: 0.0, Sum: 2.0]
      [Update RS (ms): Min: 1.7, Avg: 1.7, Max: 1.9, Diff: 0.2, Sum: 7.0]
         [Processed Buffers: Min: 5, Avg: 12.0, Max: 17, Diff: 12, Sum: 48]
      [Scan RS (ms): Min: 2.2, Avg: 2.4, Max: 2.5, Diff: 0.3, Sum: 9.7]
      [Code Root Scanning (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]
      [Object Copy (ms): Min: 5.1, Avg: 5.2, Max: 5.2, Diff: 0.1, Sum: 20.7]
      [Termination (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]
         [Termination Attempts: Min: 1, Avg: 2.0, Max: 4, Diff: 3, Sum: 8]
      [GC Worker Other (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.1]
      [GC Worker Total (ms): Min: 9.8, Avg: 9.9, Max: 9.9, Diff: 0.0, Sum: 39.4]
      [GC Worker End (ms): Min: 301333505.9, Avg: 301333505.9, Max: 301333505.9, Diff: 0.0]
   [Code Root Fixup: 0.1 ms]
   [Code Root Purge: 0.0 ms]
   [Clear CT: 0.1 ms]
   [Other: 0.5 ms]
      [Choose CSet: 0.0 ms]
      [Ref Proc: 0.1 ms]
      [Ref Enq: 0.0 ms]
      [Redirty Cards: 0.1 ms]
      [Humongous Register: 0.0 ms]
      [Humongous Reclaim: 0.0 ms]
      [Free CSet: 0.2 ms]
   [Eden: 280.0M(280.0M)->0.0B(278.0M) Survivors: 5120.0K->6144.0K Heap: 453.9M(512.0M)->174.9M(512.0M)]
 [Times: user=0.04 sys=0.00, real=0.01 secs]
```



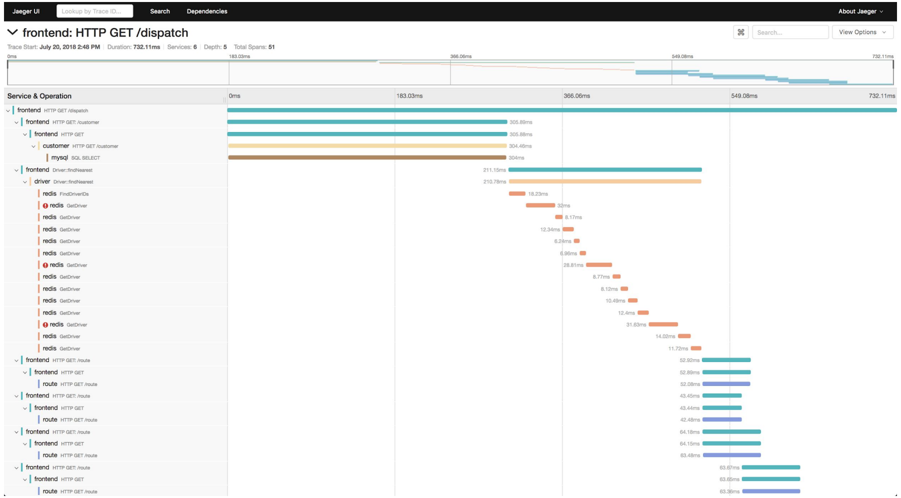
```
April 1st 2017, 03:04:21.000   message: Apr 1 03:04:21 Giriraj iTerm2[440]: Time to encode state for window <PseudoTerminal: 0x7f8f0b531a10 tabs=5 window=<PTYWindow: 0x7f8f0b654370 frame=NSRect: {{0, 0}, {1440, 900}} title=1. ./bin/logstash -f logstash.conf (java) alpha=1.000000 isMain=0 isKey=0 isVisible=1 delegate=0x7f8f0b531a10>>: 0.01567298173904419 @version: 1 @timestamp: April 1st 2017, 03:04:21.000 path: /var/log/system.log host: Giriraj.local type: syslog syslog_timestamp: Apr 1 03:04:21 syslog_hostname: Giriraj syslog_program: iTerm2 syslog_pid: 440 syslog_message: Time to encode state for window <PseudoTerminal: 0x7f8f0b531a10 tabs

April 1st 2017, 03:04:19.491   message: Apr 1 03:04:19 --- last message repeated 2 times --- @version: 1 @timestamp: April 1st 2017, 03:04:19.491 path: /var/log/system.log host: Giriraj.local type: syslog tags: _grokparsefailure syslog_severity_code: 5 syslog_facility_code: 1 syslog_facility: user-level syslog_severity: notice _id: AVsmR1uCsQnw8NxzZrsC _type: system_logs _index: syslog-2017.03.31 _score:

April 1st 2017, 03:04:19.000   message: Apr 1 03:04:19 Giriraj quicklookd[1337]: Error returned from iconservicesagent: (null) @version: 1 @timestamp: April 1st 2017, 03:04:19.000 path: /var/log/system.log host: Giriraj.local type: syslog syslog_timestamp: Apr 1 03:04:19 syslog_hostname: Giriraj syslog_program: quicklookd syslog_pid: 1337 syslog_message: Error returned from iconservicesagent: (null) received_at: April 1st 2017, 03:04:19.492 received_from: Giriraj.local syslog_severity_code: 5 syslog_facility_code: 1 syslog_facility: user-level syslog_severity: notice _id: AVsmR1uCsQnw8NxzZrsE _type: system_logs
```

- ✅ Offer direct access to the raw data iff you have a very specific idea what you are looking for already
- 🚫 Not ergonomic to query
- 🚫 Slow and hard to maintain
- 🚫 Tend to be full of a lot of noise
- 🚫 Hard to get a feel for trends

Traces - B



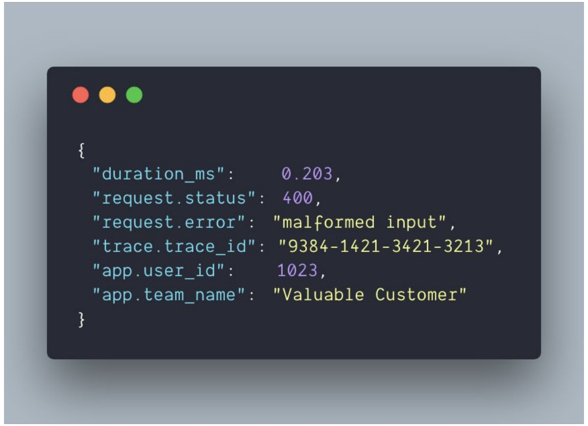
frontend: HTTP GET /dispatch

Trace Start: **July 20, 2018 2:48 PM** | Duration: **732.11ms** | Services: **6** | Depth: **5** | Total Spans: **51**

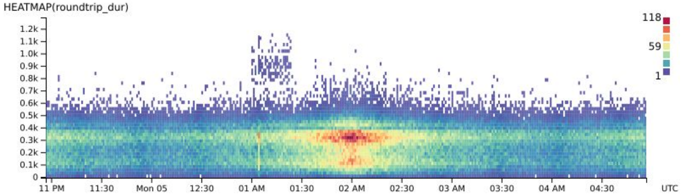
Service & Operation	0ms 183.03ms 366.06ms 549.08ms 732.11ms
frontend HTTP GET /dispatch	
frontend HTTP GET: /customer	305.89ms
frontend HTTP GET	305.88ms
customer HTTP GET /customer	304.46ms
mysql SQL SELECT	304ms
frontend Driver::findNearest	211.15ms
driver Driver::findNearest	210.78ms
redis FindDriverIDs	18.23ms
❗ redis GetDriver	32ms
redis GetDriver	8.17ms
redis GetDriver	12.34ms
redis GetDriver	6.24ms
redis GetDriver	6.96ms
❗ redis GetDriver	28.81ms
redis GetDriver	8.77ms
redis GetDriver	8.12ms
redis GetDriver	10.49ms
redis GetDriver	12.4ms
❗ redis GetDriver	31.63ms
redis GetDriver	14.02ms
redis GetDriver	11.72ms
frontend HTTP GET: /route	52.92ms
frontend HTTP GET	52.89ms
route HTTP GET /route	52.08ms
frontend HTTP GET: /route	43.45ms
frontend HTTP GET	43.44ms
route HTTP GET /route	42.48ms
frontend HTTP GET: /route	64.18ms
frontend HTTP GET	64.15ms
route HTTP GET /route	63.48ms
frontend HTTP GET: /route	63.67ms
frontend HTTP GET	63.65ms
route HTTP GET /route	63.36ms

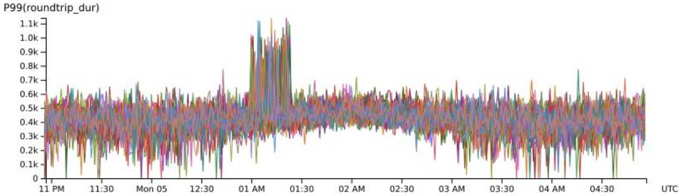
- ✅ Offer lovely view of requests as they flow through your system
- ✅ Offers access to the raw data making up every result
- 🚫 Instrumentation is hard to start small and grow outwards
- 🚫 Finding the traces you are interested in is difficult
- 🚫 Nowhere to go from problematic traces (i.e., no way to ask additional questions)

Events in Columnar Store - A



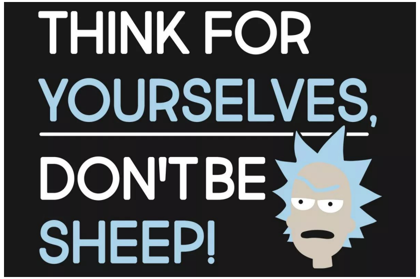
```
{
  "duration_ms":     0.203,
  "request.status":  400,
  "request.error":  "malformed input",
  "trace.trace_id": "9384-1421-3421-3213",
  "app.user_id":     1023,
  "app.team_name":  "Valuable Customer"
}
```





VENDOR DISCLAIMER

BIAS LEVELS ARE OVER 9000!!!!!!

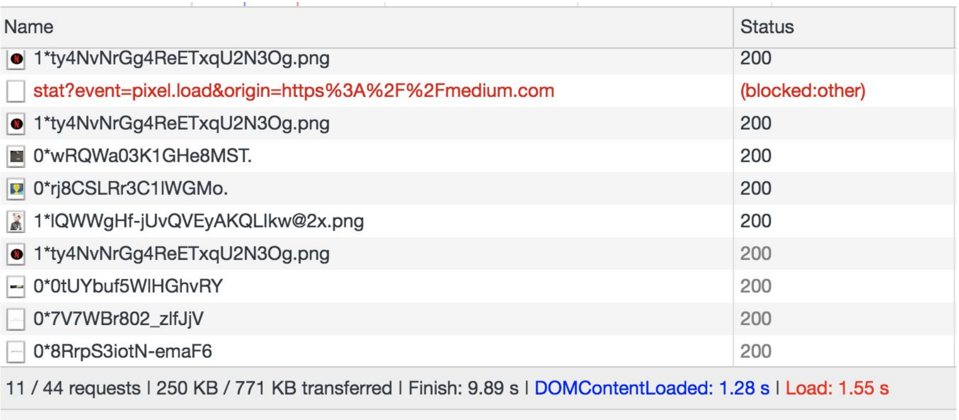


- ✅ Support for querying high cardinality fields like user ID, client version, etc.
- ✅ Offers access to the raw data making up every result
- ✅ Offers exponentially increasing VALUE when additional fields are added without exponentially increasing COST

The Health of Every Request

How many requests do most apps get per user these days?

A *FUCKLOAD*.



Name	Status
1*ty4NvNrGg4ReETxqU2N3Og.png	200
stat?event=pixel.load&origin=https%3A%2F%2Fmedium.com	(blocked:other)
1*ty4NvNrGg4ReETxqU2N3Og.png	200
0*wRQWa03K1GHe8MST.	200
0*rj8CSLRr3C1lWGMo.	200
1*lQWWgHf-jUvQVEyAKQLlkw@2x.png	200
1*ty4NvNrGg4ReETxqU2N3Og.png	200
0*0tUYbuf5WlHGhvRY	200
0*7V7WBr802_zlfJjV	200
0*8RrpS3iotN-emaF6	200

11 / 44 requests | 250 KB / 771 KB transferred | Finish: 9.89 s | DOMContentLoaded: 1.28 s | Load: 1.55 s

Everyone trashes averages, but P95 and P99 have started having dramatically less signal too.

Many of your users, not just 1/100, will hit the 99th percentile of requests.

We *need to know* context like:

- Which users or groups are seeing slowness or errors?
- Which database queries are executing slowly?
- Which hosts or containers did the problem requests pass through?
- *What specifically* is going wrong in malfunctioning background jobs?

Where we want to be

- Are all the servers running the same version?
- Which client versions are seeing errors?
- Is just one user or group seeing issues, or is everyone?
- Do we need to upgrade our instances, or fix our code?

Nope. A deploy failed halfway through and now we have two versions.

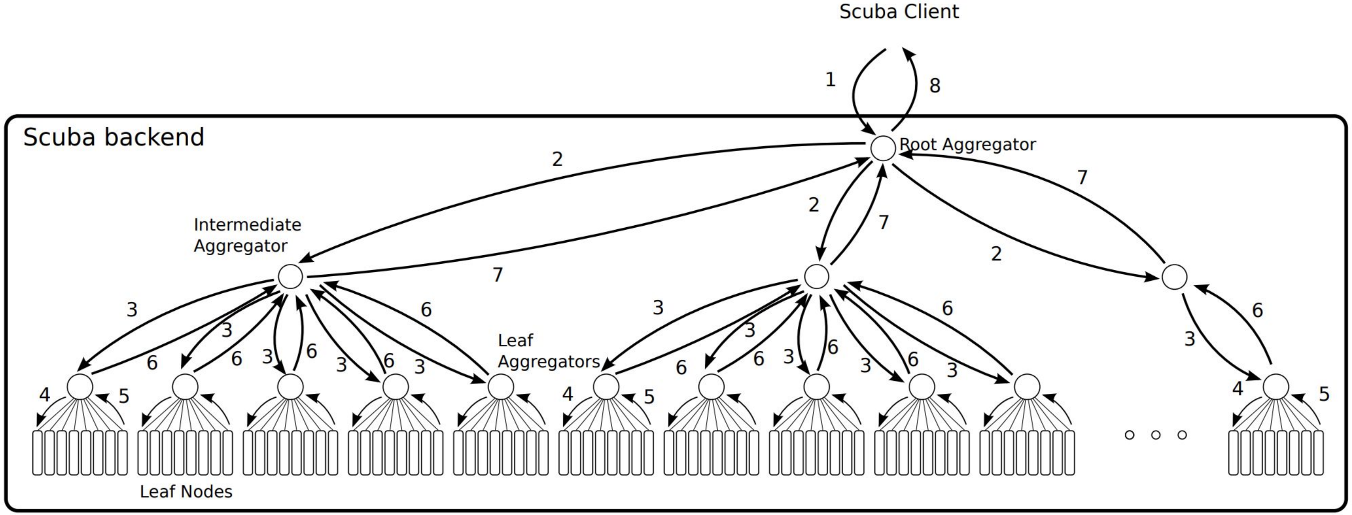
Everything lower than 2.0.1, it must have been a breaking change in our API.

It’s just one user, but they’re our biggest customer.

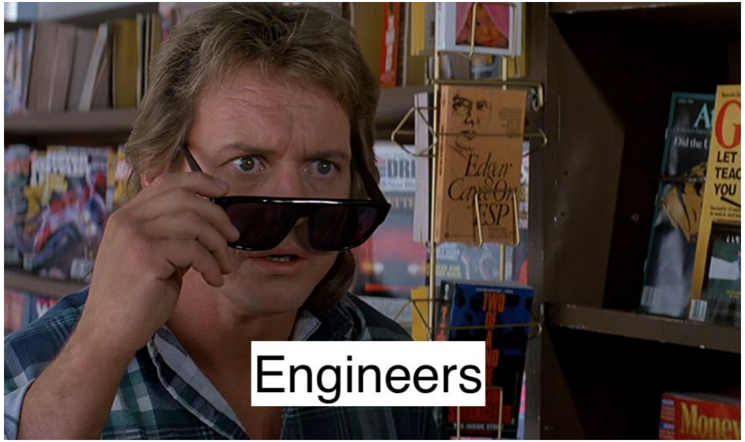
No one source of problems contributing to high CPU can be identified. Buy bigger servers.

Making o11y Affordable

Facebook pioneered SCUBA, but most of us aren't FAANG.

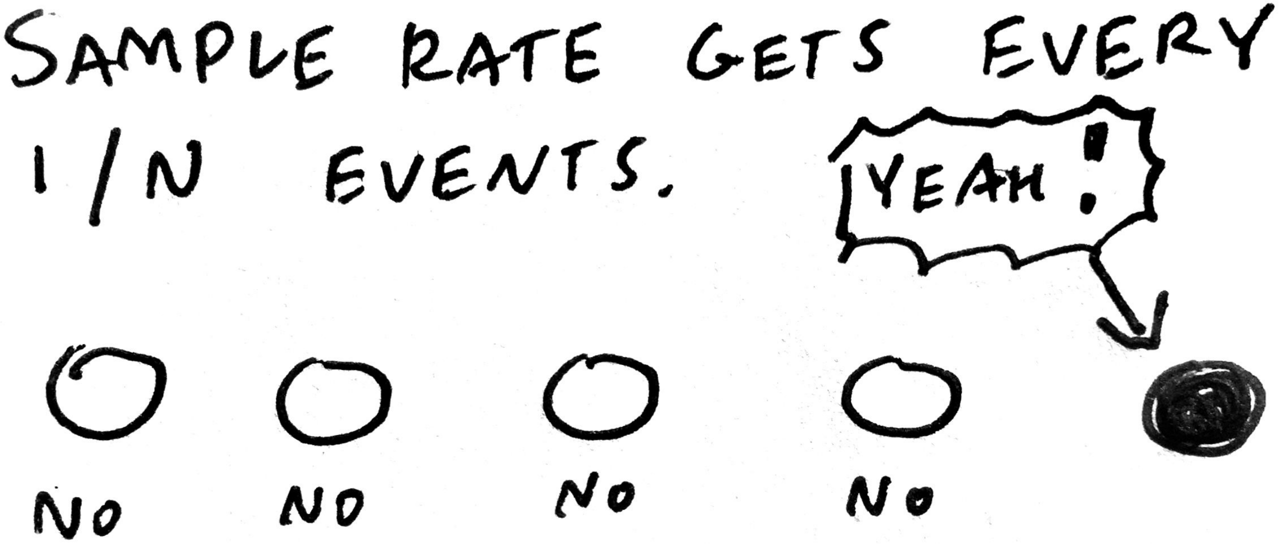


How to make o11y viable as scale increases? Sample.


Engineers

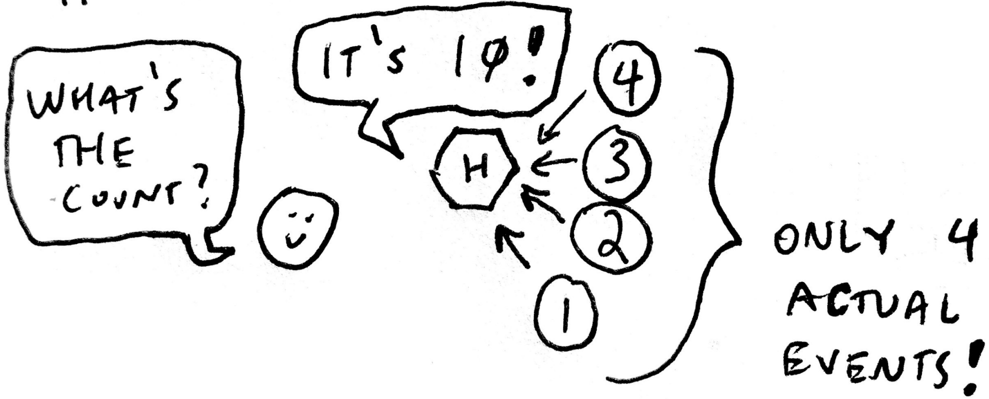
OBEY
vendors

BUY
WATCH TV
CONSUME
SLEEP
DON'T SAMPLE
SUBM
OBEY



SAMPLE RATE GETS EVERY 1/N EVENTS.

HONEYCOMB KNOWS THE SAMPLE RATE FOR EACH EVENT AND DOES THE MATH CORRECTLY EVEN IF THEY'RE UNIQUE.



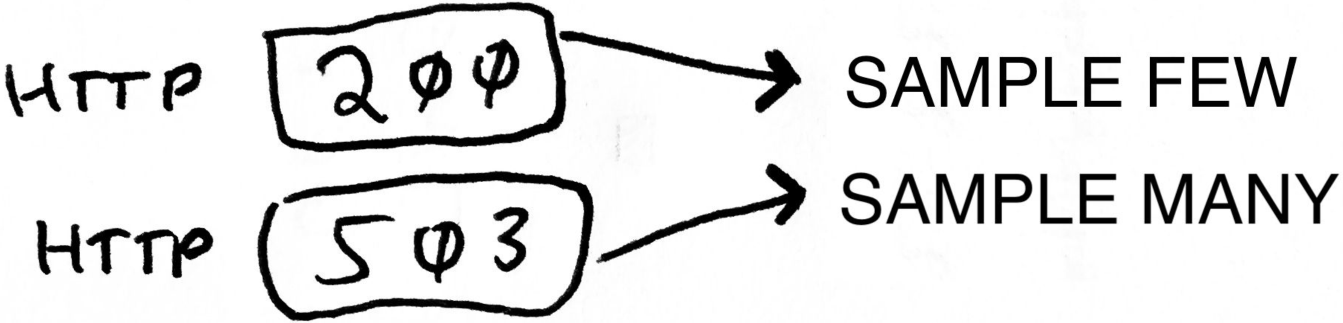
BUT THIS WHOLE TALK IS ABOUT THE HEALTH OF *EVERY* REQUEST!

OK, OK. At scale you can’t store *everything forever.*

But:

1. Statistics have your back.
2. Any problem worth worrying about will happen multiple times, or be big enough you can’t miss it.
3. Smart sampling keeps most of what you want, and less of the boring stuff.
4. In the future, we’ll likely be able to keep everything for a small duration, and sample out over time.

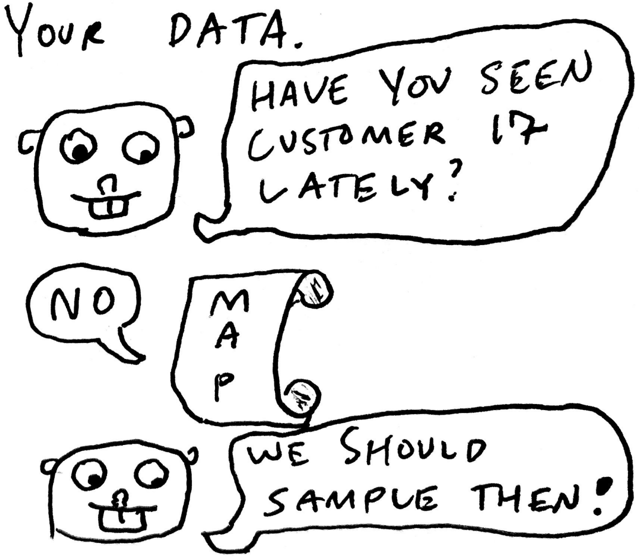
DYNAMIC SAMPLING GETS MORE OF THE GOOD STUFF AND LESS OF THE BORING STUFF.

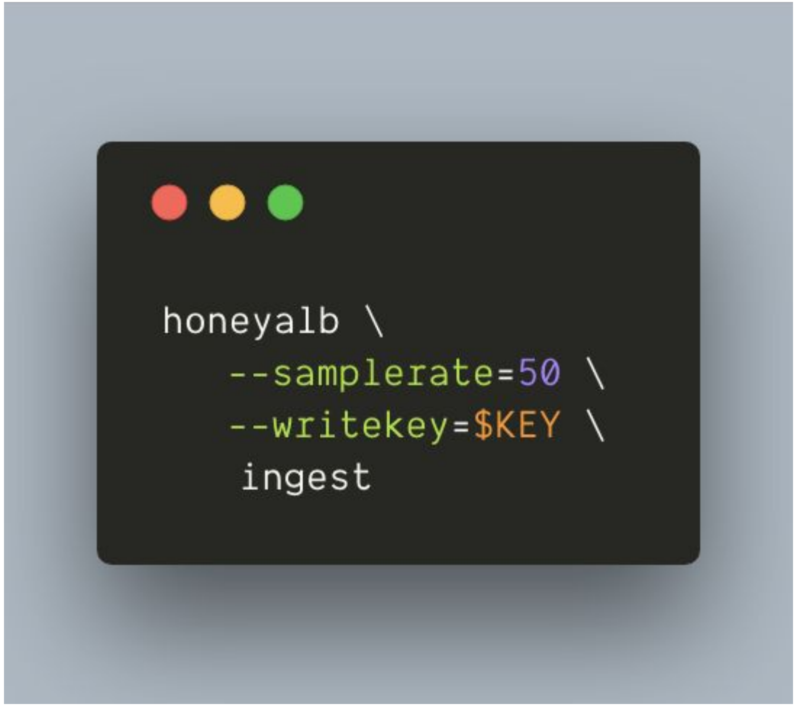


SAMPLE FEW

SAMPLE MANY

IT DOES THIS BY MAINTAINING "WEIGHTS" ASSOCIATED WITH VALUES OF A FIELD IN YOUR DATA.

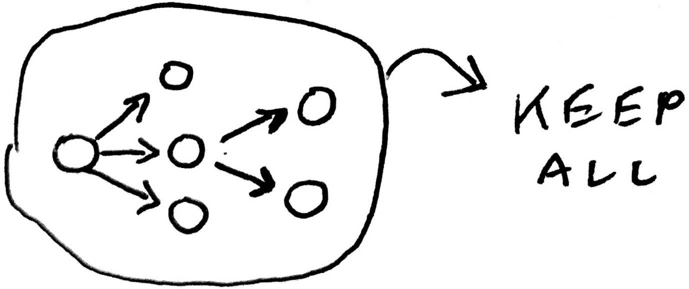


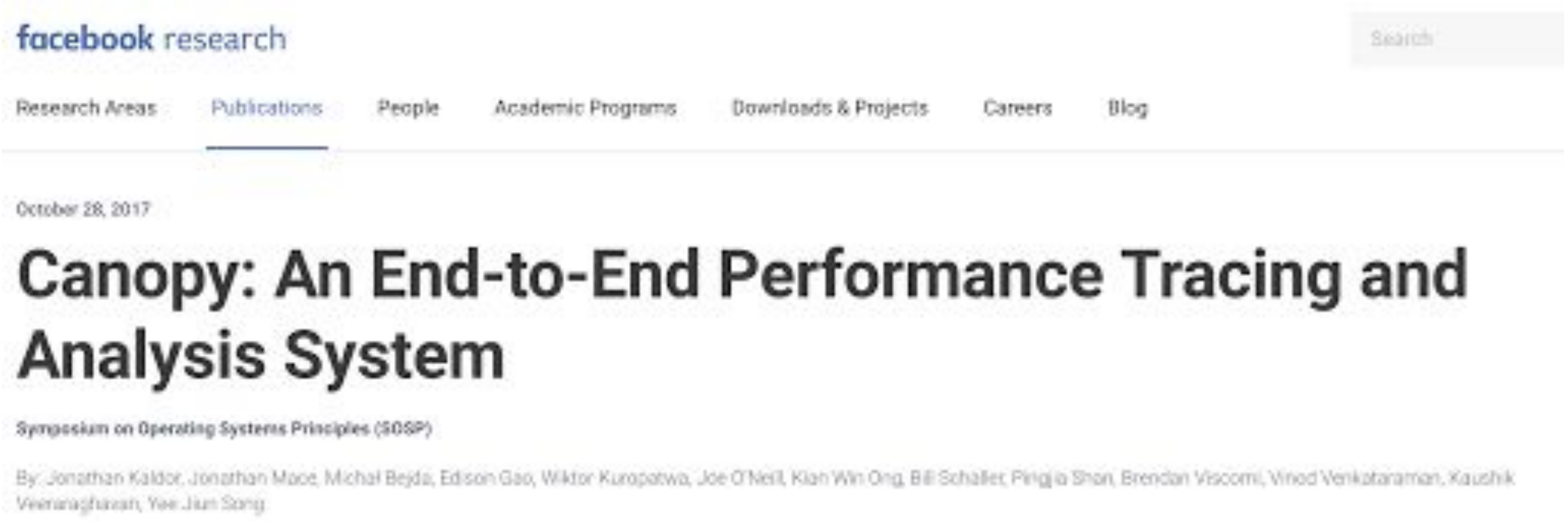


```
honeyalb \
   --samplerate=50 \
   --writekey=$KEY \
    ingest
```

Example: Crank up sample rate on ingesting Elastic Load Balancer data to 50x retention.

EVEN WHOLE TRACES CAN BE SAMPLED BY PROPAGATING A SAMPLING DECISION MADE AT THE ROOT OR BY USING DETERMINISTIC SAMPLING.





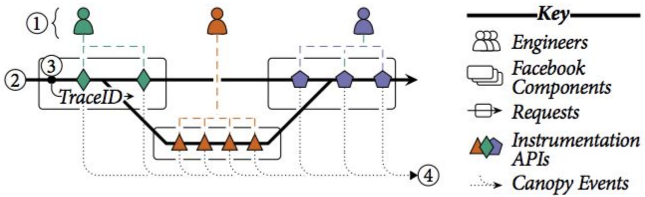
October 28, 2017

Canopy: An End-to-End Performance Tracing and Analysis System

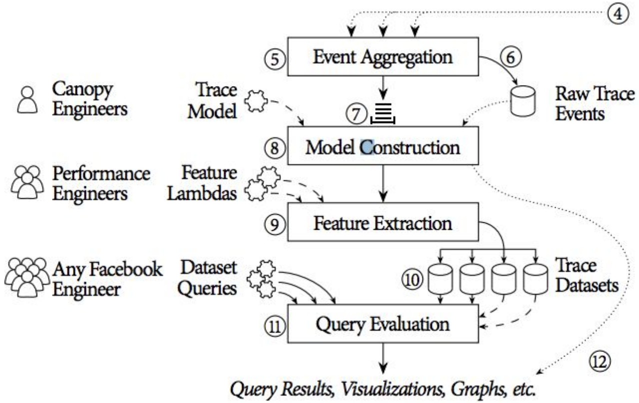
Symposium on Operating Systems Principles (SOSP)

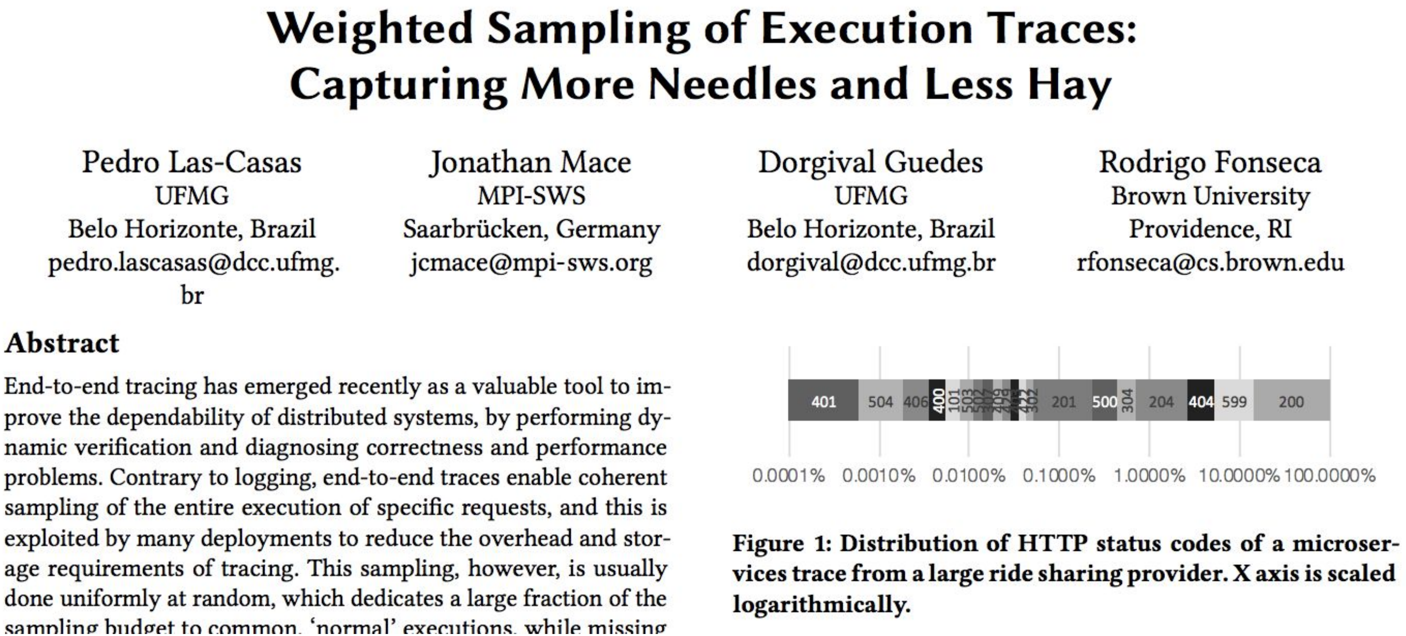
By: Jonathan Kaldor, Jonathan Mace, Michal Bejda, Edison Gao, Wiktor Kuropatwa, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, Vinod Venkataraman, Kaushik Veeraraghavan, Yee Jiun Song

https://research.fb.com/publications/canopy-end-to-end-performance-tracing-at-scale/



(a) Engineers instrument Facebook components using a range of different Canopy instrumentation APIs (①). At runtime, requests traverse components (②) and propagate a TraceID (③); when requests trigger instrumentation, Canopy generates and emits events (④).





Weighted Sampling of Execution Traces: Capturing More Needles and Less Hay

Pedro Las-Casas
UFMG
Belo Horizonte, Brazil
pedro.lascasas@dcc.ufmg.br

Jonathan Mace
MPI-SWS
Saarbrücken, Germany
jcmace@mpi-sws.org

Dorgival Guedes
UFMG
Belo Horizonte, Brazil
dorgival@dcc.ufmg.br

Rodrigo Fonseca
Brown University
Providence, RI
rfonseca@cs.brown.edu

Abstract

End-to-end tracing has emerged recently as a valuable tool to improve the dependability of distributed systems, by performing dynamic verification and diagnosing correctness and performance problems. Contrary to logging, end-to-end traces enable coherent sampling of the entire execution of specific requests, and this is exploited by many deployments to reduce the overhead and storage requirements of tracing. This sampling, however, is usually done uniformly at random, which dedicates a large fraction of the sampling budget to common, 'normal' executions, while missing

Figure 1: Distribution of HTTP status codes of a microservices trace from a large ride sharing provider. X axis is scaled logarithmically.

https://people.mpi-sws.org/~jcmace/papers/lascasas2018weighted.pdf

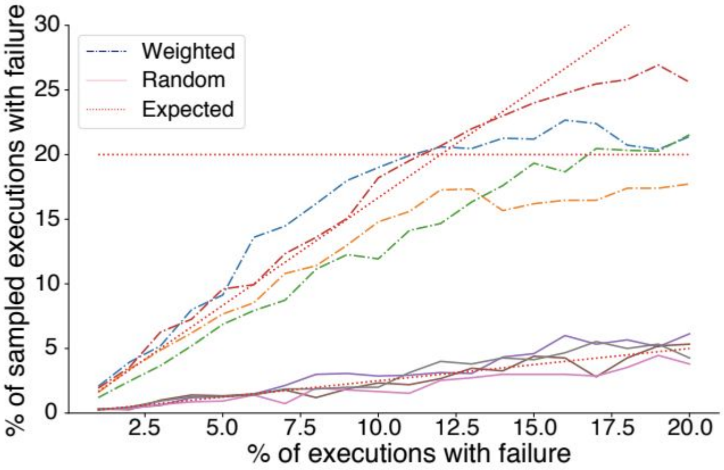
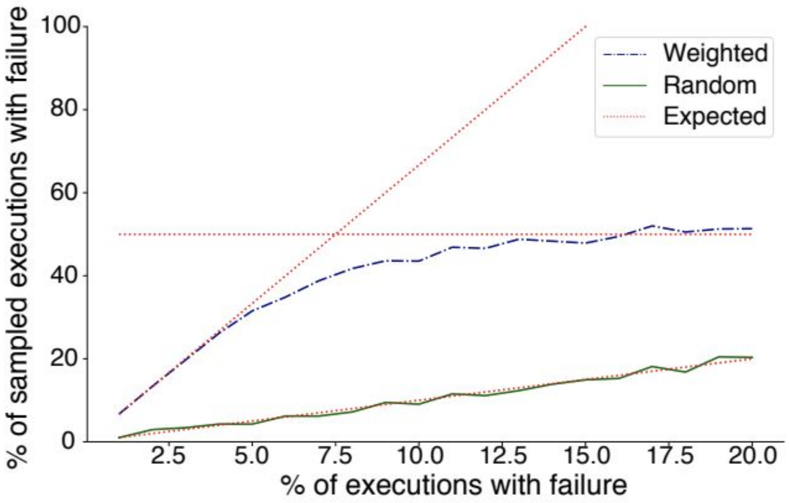


Figure 5: Percentage of failed traces (VM Fail) in the selected sample.



Key Takeaways

- Observability gets you answers about the “why”, “how”, “what” of issues that monitoring cannot and can reduce issue resolution time from days to minutes.
- Sampling is a great way to make o11y affordable and scalable.
- **Observability will be a key differentiator in successful businesses in the coming years.**

Thanks for coming to my talk !

I’m on Twitter -

@dotpem

E-mail me:

nathan@honeycomb.io

Or come talk to me at our booth!