Amber Race

Senior SDET

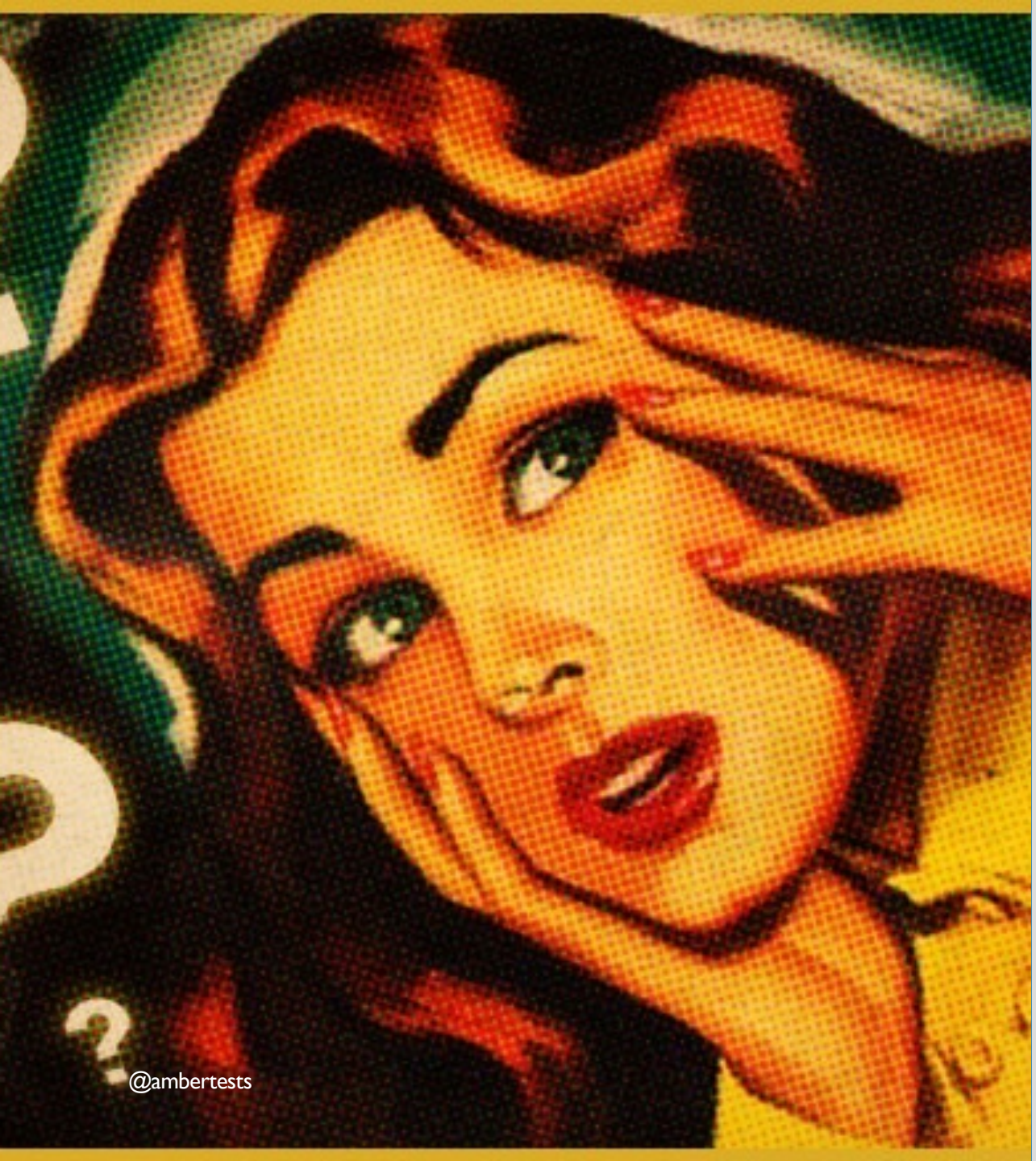
Big Fish Games

@ambertests

THE JOY OF TESTING IN PRODUCTION

ONCE UPON A TIME THERE WAS A SERVICE…

- In production since 2014
- Weird interface
- Built to support multiple games
- Rich set of APIs
- Currently serving 800k DAU, 4000 requests per second



WHO? WHAT? WHY?

- Didn't have detailed API use profiles
- Didn't know how often API calls were failing
- Didn't know range of response times
- Sometimes requests would get backed up and we didn't know why

THE MISSING PIECE

What we had

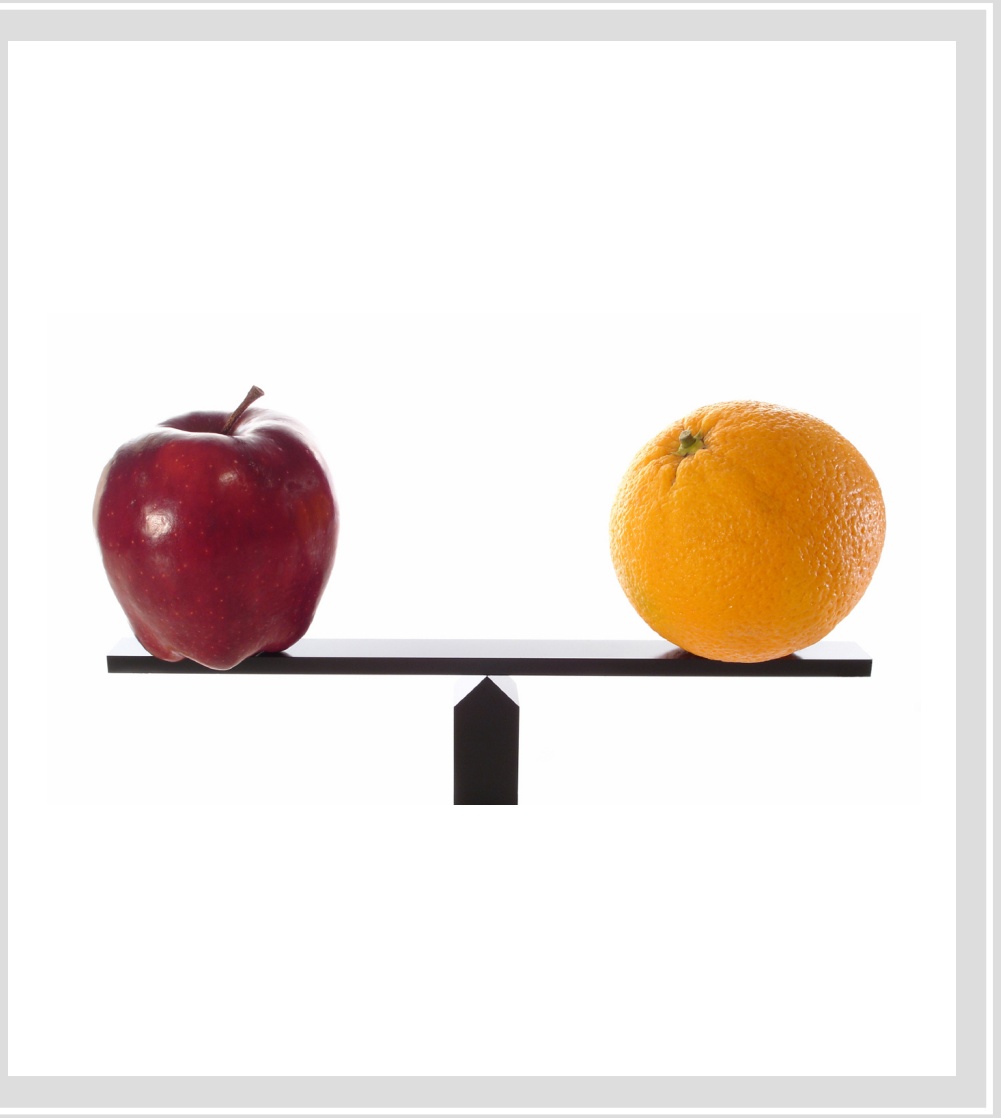
- Unit and integration tests
- Continuous build and deployment
- Multiple test environments
- Extensive manual testing of games

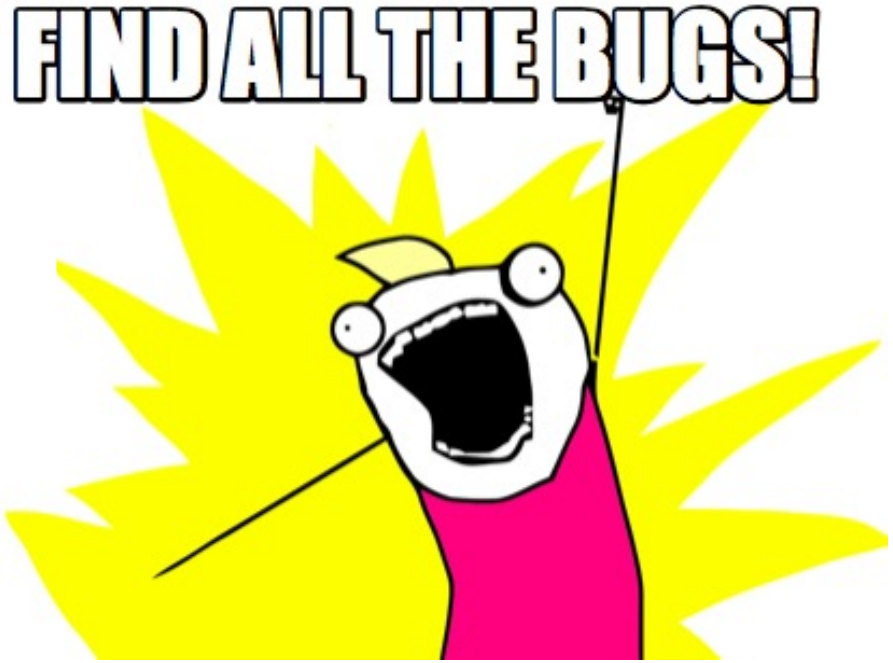
What we didn't have

- A view of what was really happening in production

BUT ISN'T TESTING IN PRODUCTION BAD??

YOU ARE ALREADY TESTING IN PRODUCTION





TO AVOID "TESTING IN PRODUCTION", YOU HAVE TO FIND ALL THE BUGS BEFORE YOU RELEASE

THERE WILL ALWAYS BE ONE THAT GETS AWAY…

PROBLEMS WITH THE SANDBOX

- Difficult to cover all client combinations
  - Browsers
  - Devices
  - Networks
- Does your test environment match production?
  - Memory
  - CPU
  - DB config
- Does your test user base match production?

  -> No, it does not

Basic Functional Testing

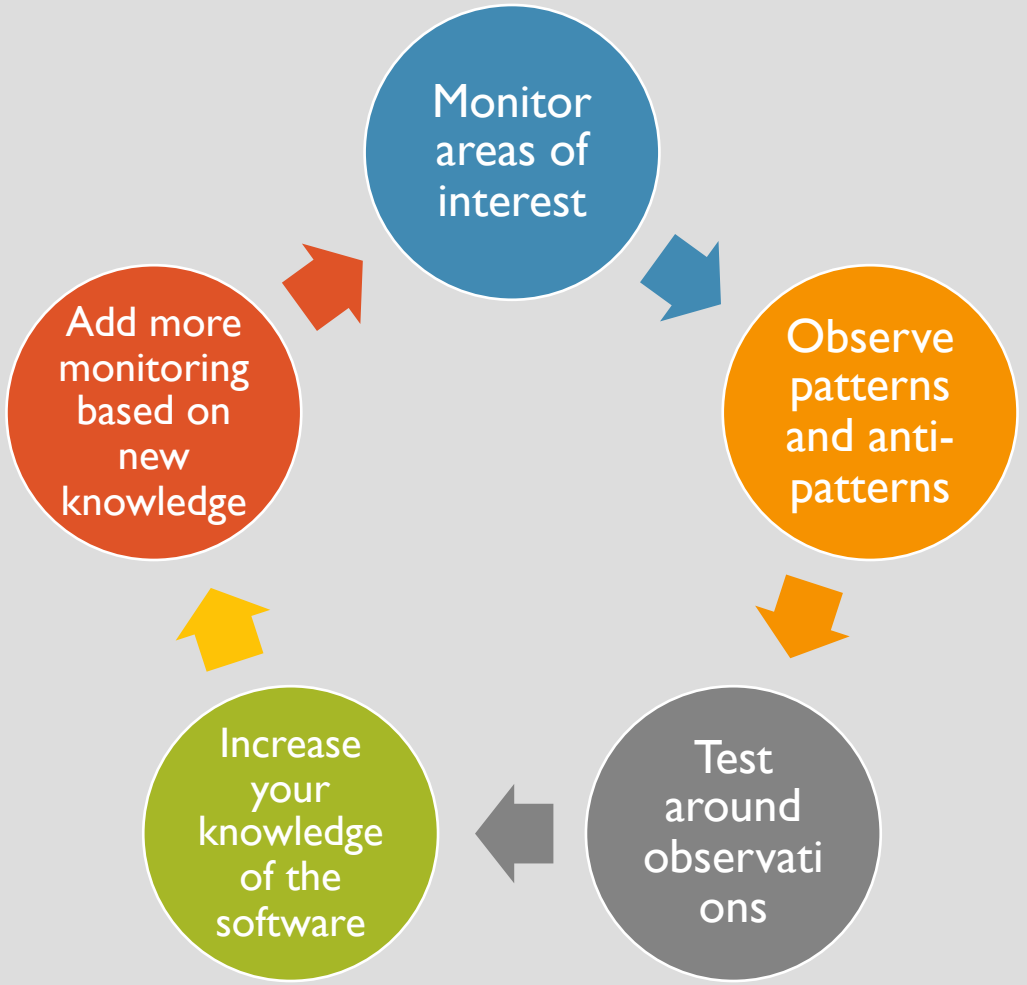
- Happy paths tested
- Basic regression passes
- Boundary cases covered

Thorough Test Coverage

- Full suite of unit tests
- Continuous build with automated tests
- Performance and load testing
- Security test pass
- Exploratory sessions

The Real World

HOW PRODUCTION MONITORING HELPS YOUR TESTING



COMMON MONITORING TOOLS

LOG-BASED

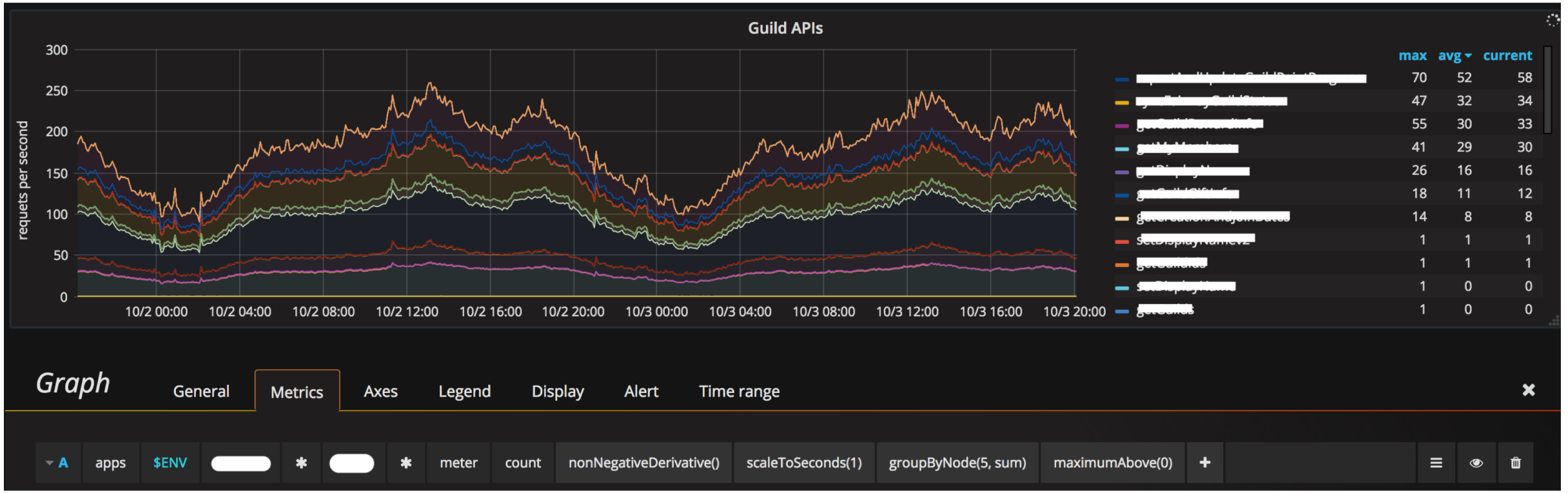
- ELK Stack (Elasticsearch + Logstash + Kibana)
- Splunk

DB-BASED

- Graphite
- StatsD
- InfluxDB

DISPLAY

- Grafana
- Kibana



Graphite + Grafana

WHY DID THIS WORK FOR US?

Stack supported by Ops

Instrumenting the code was easier than fixing the logs

Graphite query language gives a lot of flexibility

Grafana templating is very useful

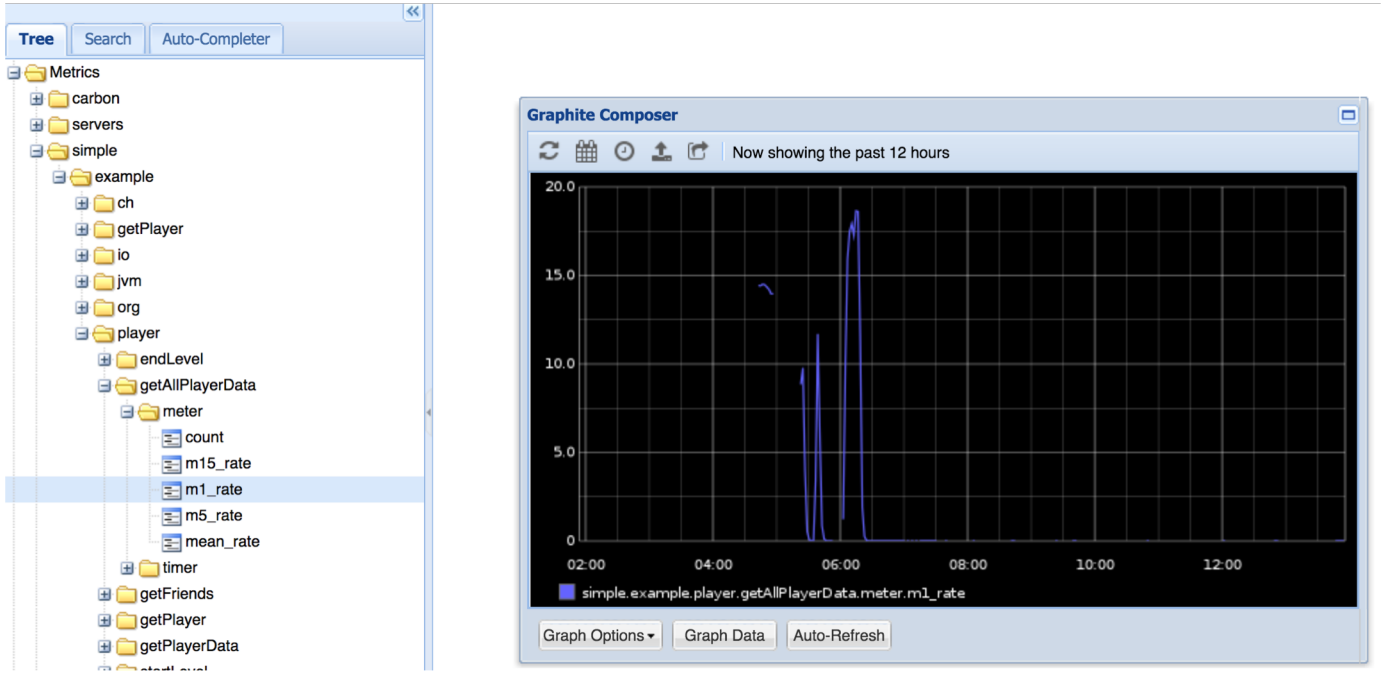
Pretty graphs are nice to look at

```
public static void recordServiceMetrics(
    String metricName, boolean success, long responseTime, String exception) {

  String requestCounterName =
      String.format("Services.%s.%s.RequestCount", ServerMain.HostName, metricName);
  String failCounterName =
      String.format("Services.%s.%s.FailReplies", ServerMain.HostName, metricName);
  String responseTimerName =
      String.format("Services.%s.%s.ResponseTime", ServerMain.HostName, metricName);
  String exceptionCounterName =
      String.format("Services.%s.%s.Exceptions.%s", ServerMain.HostName, metricName, exception);

  registerMetrics(
      requestCounterName,
      failCounterName,
      responseTimerName,
      exceptionCounterName,
      success,
      responseTime,
      exception);
}
```

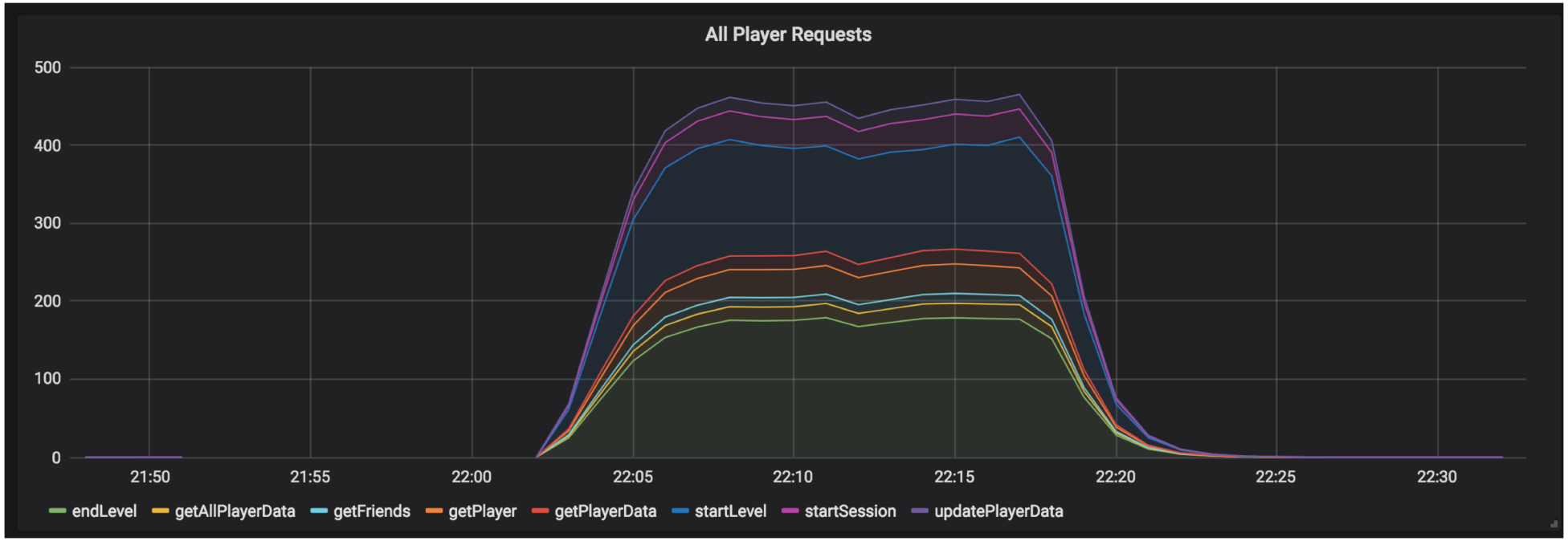
INSTRUMENTING THE CODE

```java
@GET
@UnitOfWork
@Path("/{playerId}")
@Timed(name="player.getPlayer.timer", absolute=true)
@Metered(name="player.getPlayer.meter", absolute=true)
@ExceptionMetered(name="player.getPlayer.errors", absolute=true)
public Player getPlayer(@PathParam("playerId") LongParam playerId) {
    return findSafely(playerId.get());
}
```

INSTRUMENTING THE CODE



STORAGE IN GRAPHITE



VIEW IN GRAFANA

SAMPLE WALKTHROUGH

WHAT DO WE TRACK?

- Specific APIs
  - Request Count
  - Failures
  - Response time
- Memcache Usage
- Thread Counts per Machine
- Exceptions Thrown
- Third Party Service Response Time
- System Stats (CPU/Memory/Ports)
- Database Stats (Reads, Writes, Slow Queries

WHAT WE FOUND

@ambertests

POOR API USAGE

- Certain API calls failed every time because the client was making an invalid call
- Save APIs called too frequently

MEMCACHE ISSUES

- One API was attempting to save a null value, causing lots of unnecessary traffic

DATABASE OVERUSE

- In one game, 90% of API calls were to save data because it was saving too frequently
- Another very common call was hitting database when 95% of time nothing was there

SLOW REQUESTS

- By monitoring third party calls and breaking down individual API response, we were able to correlate stoppage issues with third party network issues

THREAD CONFIG ISSUES

- Tracking worker thread creation in the service exposed issues with thread pools that were too small

DASHBOARD MAINTENANCE

- Make sure you only have the things you care about
- Consolidate as much as possible
- Check that your solution isn't adding too much overhead
- Re-evaluate your metrics

MONITORING → OBSERVABILITY

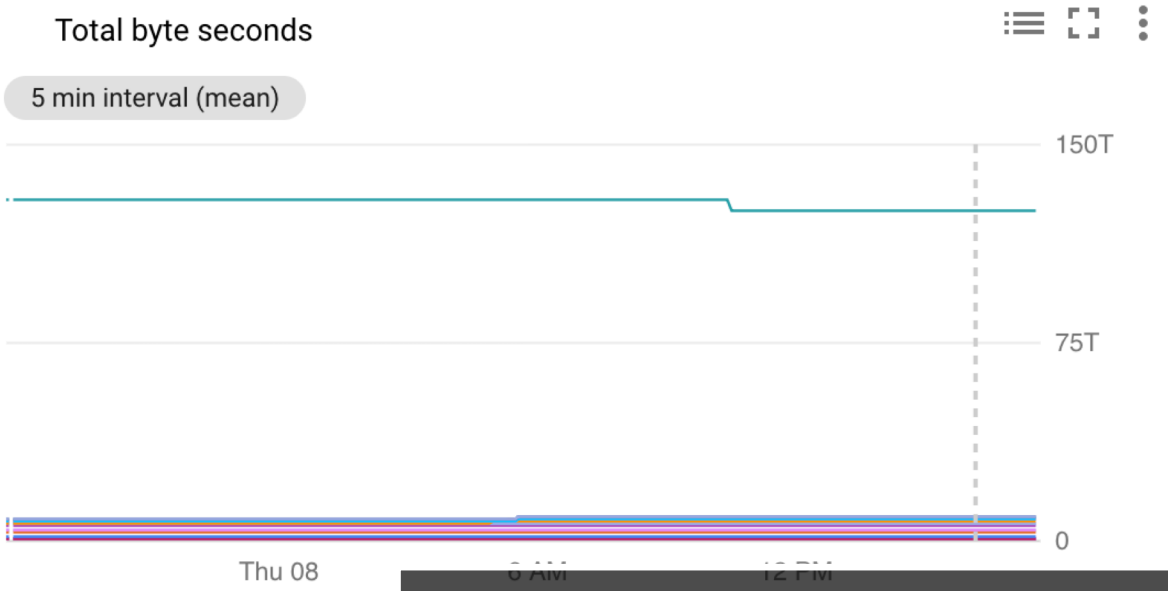
Store complete data, not just aggregates

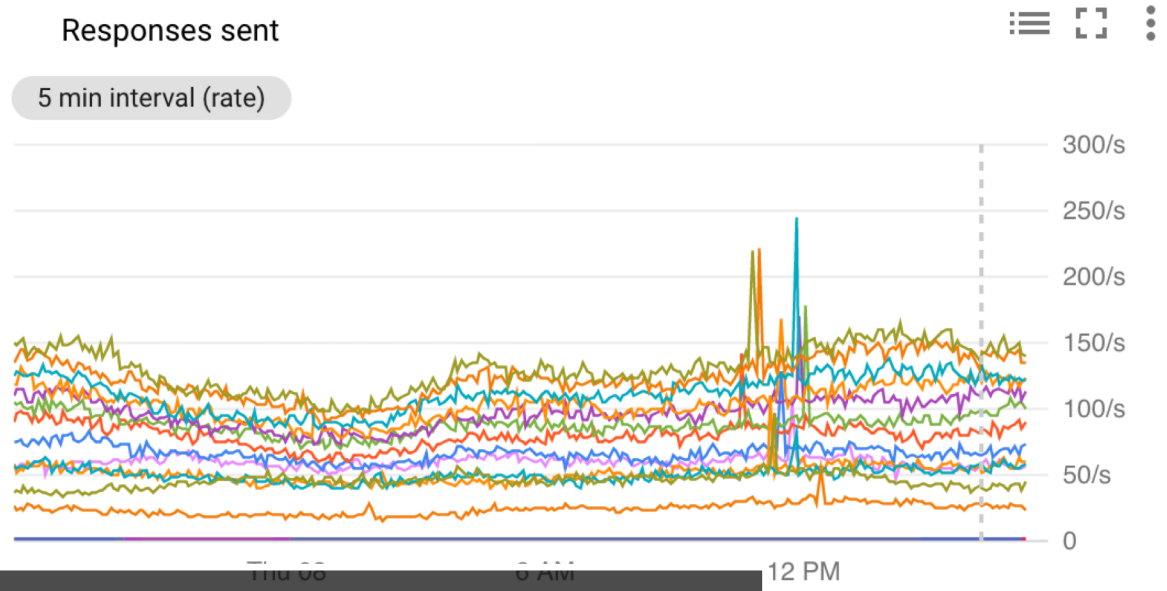
Ability to drill down to specific events

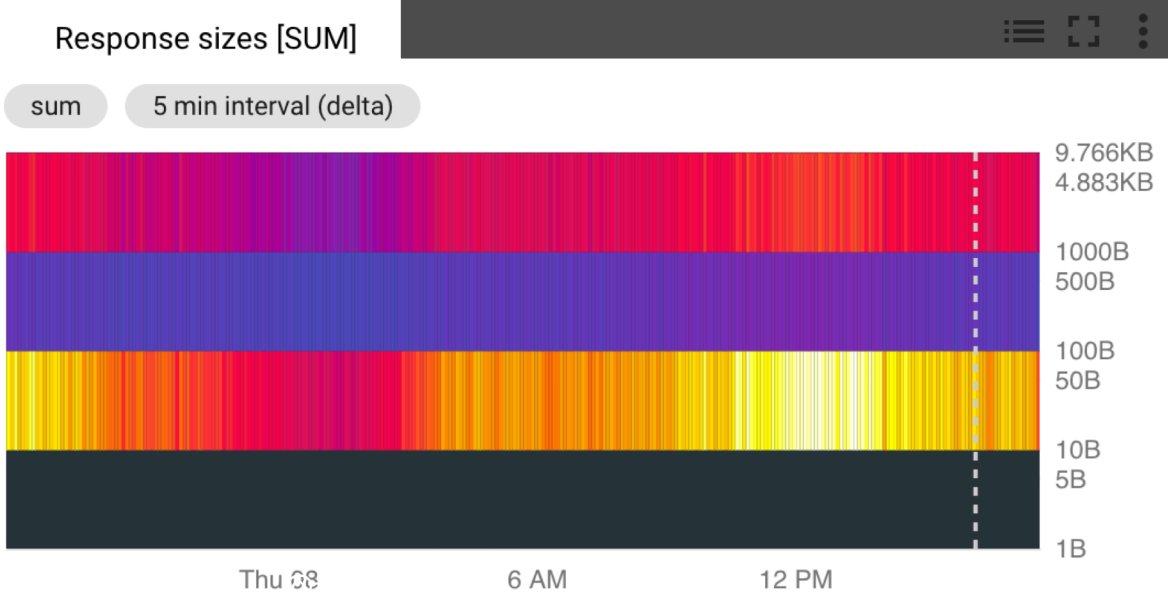
Tying together disparate info spread across multiple dashboards

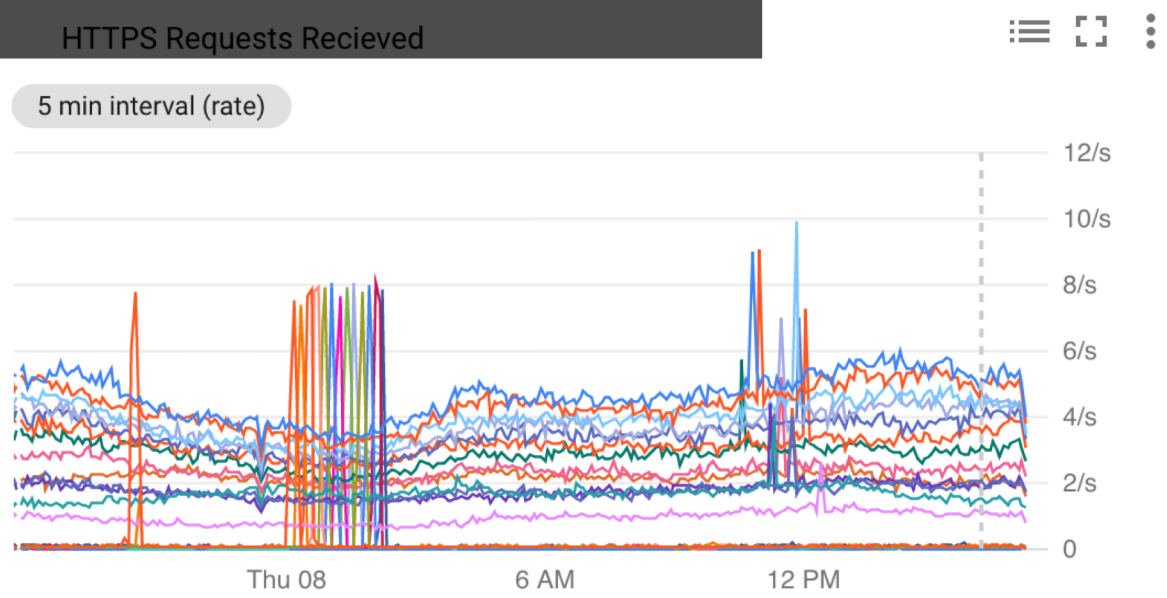
Follow Charity Majors (@mipsytipsy) of honeycomb.io for more

TYPICAL METRICS DASHBOARD

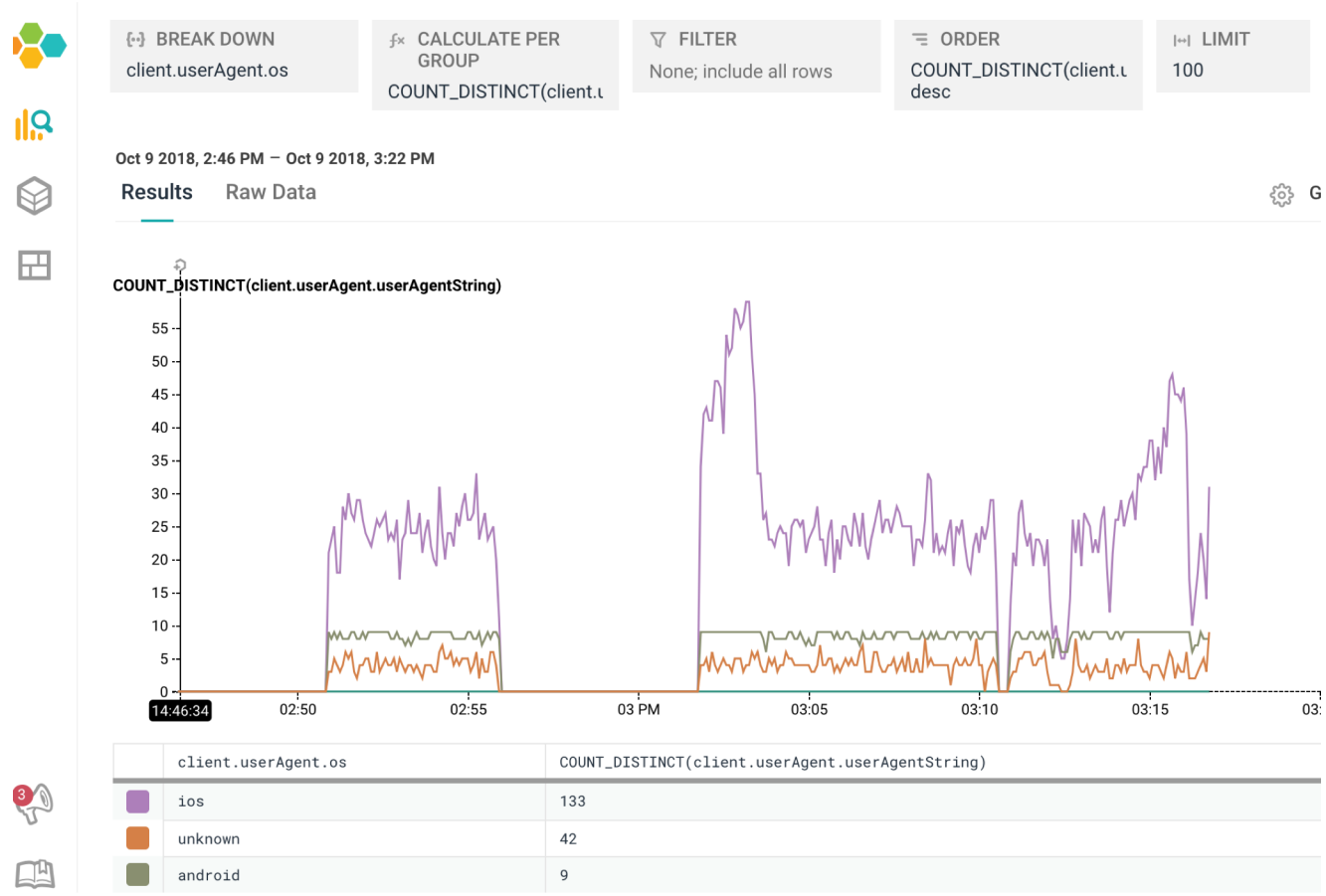






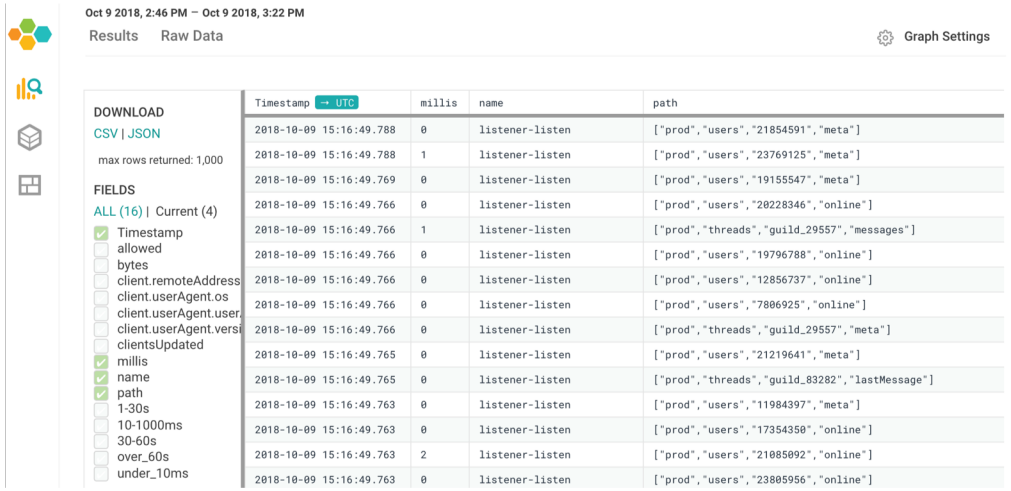


WITH ALL THE DATA, YOU CAN AGGREGATE...



@ambertests

..AND THEN ZOOM IN

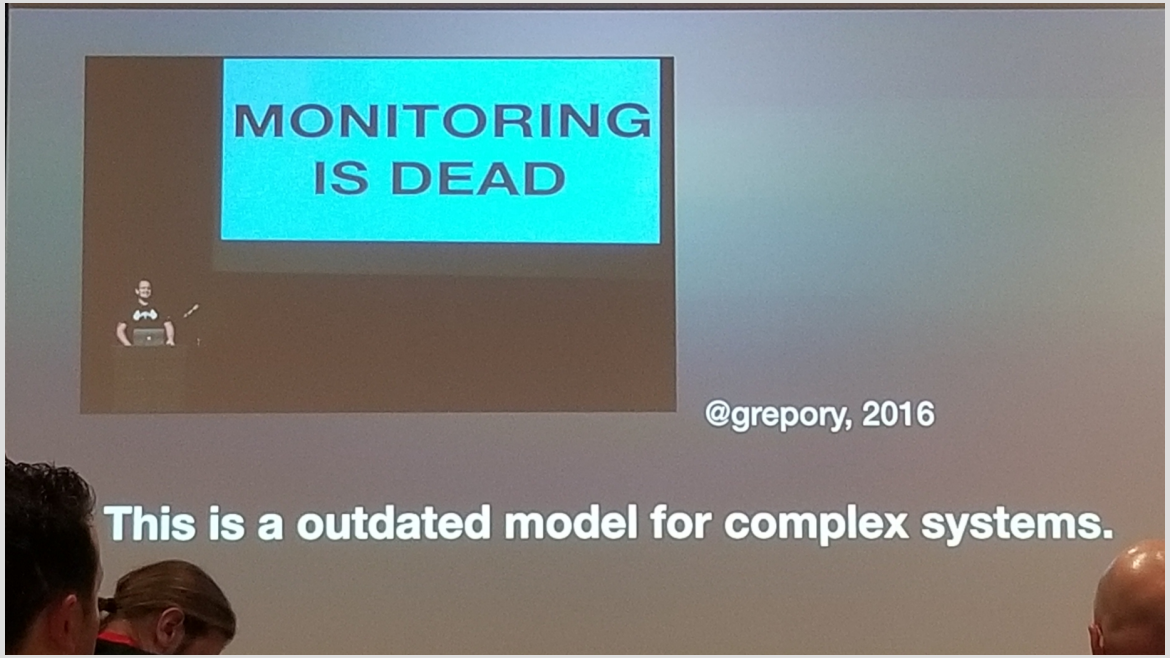


89.24.[redacted]	["prod","users","20228346","online"]
89.24.[redacted]	["prod","threads","guild_29557","messages"]
89.24.[redacted]	["prod","users","19796788","online"]
89.24.[redacted]	["prod","users","12856737","online"]
89.24.[redacted]	["prod","users","7806925","online"]
89.24.[redacted]	["prod","threads","guild_29557","meta"]

HONEYCOMB WALKTHROUGH

SO WHY BOTHER WITH TRADITIONAL MONITORING?

- Monitoring tools are easily available, open source
- Dashboards are still useful as a high-level view
- Monitoring tools can still inform further testing
- Some information is better than no information



Slide credit: Charity Majors (@mipsytipsy)

BUT THAT'S ALL JUST DEBUGGING. WHAT ABOUT TESTING???

PERFORMANCE TESTING

- New APIs can be on production servers before clients are updated
- No need to translate results between environments
- Test will automatically include real background load

FEATURE TESTING

- Flag new features at the config level to toggle on and off
- Update configuration on select boxes
- Watch what happens

CHAOS TESTING

MONITORING, OBSERVING, AND TESTING GO TOGETHER

Martin (马丁) Hynie
@vds4

Following

An attempt to paraphrase a useful analogy from @mipsytipsy, but in tester speak... consider monitoring like your automated checks (known unknowns) while observability is building and enabling exploratory testing in prod (unknown unknowns). #testinprod

7:24 AM - 8 Nov 2018

"Tester's **don't break** products, just illusions people have about them."–
Maaret Pyhäjärvi (@maaretp)

EXPLORING IS PART OF TESTING


EXPLORE
WITHOUT
FEAR
@ambertests

MORE INFORMATION

- ELK Stack: https://www.elastic.co/elk-stack
- Graphite + StatsD Docker image: https://hub.docker.com/r/hopsoft/graphite-statsd/
- Graphite Docs: https://graphite.readthedocs.io/en/latest/
- StatsD: https://github.com/etsy/statsd
- Grafana: https://grafana.com/
- Honeycomb: https://www.honeycomb.io/

CONTACT ME!

Amber Race

Twitter: @ambertests

LinkedIn: https://www.linkedin.com/in/amber-race-tests/

GitHub: https://github.com/ambertests