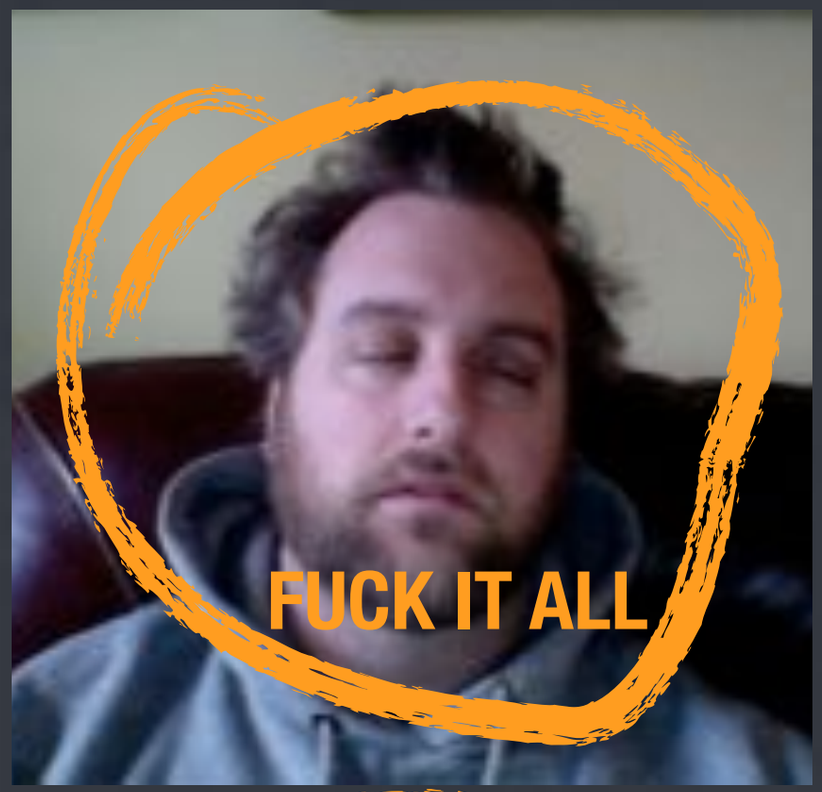
Monitoring & Observability

Getting off the starting blocks.



THE MANY FACES OF THEO

FUN WITH BEARDS AND HAIR

Agenda

- Define stuff.
- Set some tenets.
- Discuss and implement some tenets.
- Answer a lot of questions.

Monitoring... what it is.

- We'll get to that.

Observability

- Being able to measure “things” or witness state changes.
- Not useful if doing so alters behavior (significantly).

Development & Production

- For the rest of this talk...

  There is only production.

Data & Information Terms

- Measurement: a single measurement of something
  - a value on which numerical operations make sense:
    - 1, -110, 1.234^{123}, 9.886^{-19}, 0, null
    - “200”, “304”, “v1.234”, “happy”, null

Data & Information Terms

- Metric: something that you are measuring
  - The version of deployed code
  - Total cost on Amazon services
  - total bugs filed, bug backlog
  - Total queries executed

Notice no rates

- DO NOT STORE RATES.

Measurement Velocity

- The rate of change of measurements.

Perspective

- Sometimes perspective matters
  - page load times, DNS queries,
  - consider RUM (real user monitoring)
- Usually it does not
  - total requests made against a web server

Visualization

- The assimilation of multiple measurements into a visual representation.

Trending

- Understanding the "direction" of series of measurements on a metric.
- Here direction is loose and means "pattern within."

Alerting

- To bring something to one's attention.

Anomaly Detection

- The determination that a specific measurement is not within reason.

Monitoring... what it is.

- All of that.

Review

- Measurement
- Measurement Velocity
- Metric
- Perspective
- Visualization
- Trending
- Alerting
- Anomaly Detection
- Observability
- Monitoring

Some Tenets

- Most people suck at monitoring.
- They monitor all the wrong things (somewhat bad)
- The don’t monitor the important things (awful)

Do not collect rates of things

- Rates are like trees making sounds falling in the forest.
- Direct measurement of rates leads to data loss and ultimately ignorance.

Prefer high level telemetry

1. Business drivers via KPIs,
2. Team KPIs,
3. Staff KPIs,
4. ... then telemetry from everything else.

Implementation

- Herein it gets tricky.

Only because of the tools.

- I could show you how to use tool X, or Y or Z.
- But I wrote Reconnoiter and founded Circonus because X, Y and Z didn't meet my needs.
- Reconnoiter is open.
- Circonus is a service.

Methodology

- I’m going to focus on methodology
  that can be applied across whatever toolset you have.

Pull vs. Push

- Anyone who says one is better than the other is... WRONG.
- They both have their uses.

Reasons for pull

1. Synthesized observation is desirable.
2. Observable activity is infrequent.
3. Alterations in observation frequency are useful.

Reasons for push

- Direct observation is desirable.
- Discrete observed actions are useful.
- Discrete observed actions are frequent.

False reasons.

- Polling doesn't scale.

Protocol Soup

- The great thing about standards is... there are so many to choose from.

Protocol Soup

- SNMP(v1,v2,v3) both push(trap) and pull(query)
- collectd(v4,v5) push only
- statsd push only
- JMX, JDBC, ICMP, DHCP, NTP, SSH, TCP, UDP, barf.

Color me RESTy

- Use JSON.
- HTTP(s) PUT/POST somewhere for push
- HTTP(s) GET something for pull

High-volume Data

- Occasionally, data velocity is beyond what's reasonable for individual HTTP PUT/POST for each observation.

1. You can fall back to UDP (try statsd)
2. I prefer to batch them and continue to use REST

nad

- nad is great. use nad.
- https://github.com/circonus-labs/nad
- Think of it like an SNMP that's
  - actually Simple
  - Monitoring **not Management**
  - and trivial extended to suit your needs

nad online example

To the Internet ➥

But wait...

- nad isn't methodology...
- it's technology.

Correct...

- Back to the topic.
- I talked about nad briefly to provide a **super simple** tool to erase the question: “but how?”

The real question is: “what?”

- What should I be monitoring?
- This is the best question you can ask yourself.
  - Before you start.
  - While you’re implementing.
  - After you’re done.

The industry answer:

- MONITOR ALL THE THINGS!
- I'll tell you this too, in fact.
- But we have put the cart ahead of the horse.

Question?

- If I could monitor one thing, what would it be?

  hint: CPU utilization on your web server ain't it.

Answer:

- It depends on your business.
- If you don’t know the answer to this,
  I suggest you stop worrying about monitoring
  and start worrying about WTF your company does.

Here, we can’t continue.

- Unless I make stuff up...
- So, here I go makin’ stuff up.

Let us assume

- we run a web site where customers buy products

Monitoring purchases.

- So, we should monitor how many purchases were made and ensure it is within acceptable levels.

- Not so fast.

Actually.

- We want to make sure customers **can** purchase from the site and **are** purchasing from the site.
- This semantic different is critically important.
- And choosing which comes down to velocity.

What is this velocity thing?

- Displacement / time
  (i.e. purchases/second or $/second)
- BUT WAIT! You said:
  “Do not collect rates of things.”
- Correct...
  collect the displacement,
  visualize and alert on the rate.

So which?

- High velocity w/ predictably smooth trends: velocity is more important
- Low velocity or uneven arrival rates: measuring capability is more important

To rephrase

- If you have sufficient real data,
  observing that data works best;
- otherwise, you must
  synthesize data and monitor that.

As a tenet.

- Always synthesize.
- **additionally** observe real data when possible

More demonstrable (in a short session)

- I've got a web site that my customers need to visit.
- The business understands that we need to serve customers with at least a basic level of QoS: no page loads over 4s

Active checks.

A first attempt

- curl http://surge.omniti.com/
- extract the HTTP response code
- if 200, we're super good!
- Admittedly not so good.

A wealth of data.

- Synthesizing an HTTPS GET could provide:
  - SSL Subject, validity, expiration
  - HTTP code, Headers and Content
  - Timings on TCP connection, first byte, full payload

Still, this is highly imperfect.

- Don't get me wrong, they are useful.
  We use them all over the place... they are **cheap**.
- But, ideally, you want to load the page closer to the way a user does (all assets, javascript, etc.)
- Enter phantomjs

```
var page = require('webpage').create();
page.viewportSize = { width: 1024, height: 768 };

page.onError = function(err) { stats.errors++; };
page.onInitialized =
   function() { start = new Date(); };
page.onLoadStarted =
  function() { stats.load_started = new Date() - start; };
page.onLoadFinished =
  function() { stats.load_finished = new Date() - start; };
page.onResourceRequested = function() { stats.res++; };
page.onResourceError = function(err) { stats.res_errors++; };
page.onUrlChanged = function() { stats.url_redirects++; };

page.open('http://surge.omniti.com/', function(status) {
    stats.status = status;
    stats.duration = new Date() - start;
    console.log(JSON.stringify(stats));
    phantom.exit();
});
```

```
var start, stats = {
  status: null
, errors: 0
, load_started: null
, load_finished: null
, resources: 0
, resource_errors: 0
, url_redirects: 0
};
```

Passive checks.

Now for the passive stuff

- Some examples are Google Analytics, Omniture, etc.
- Statsd (out-of-the-box) and Metrics are mediocre approach.
- If we have a lot of observable data N,
  $\bar{N}$ isn't so useful,
  σ, |N|, q(0.5), q(0.95), q(0.99), q(0), q(1), add a lot.

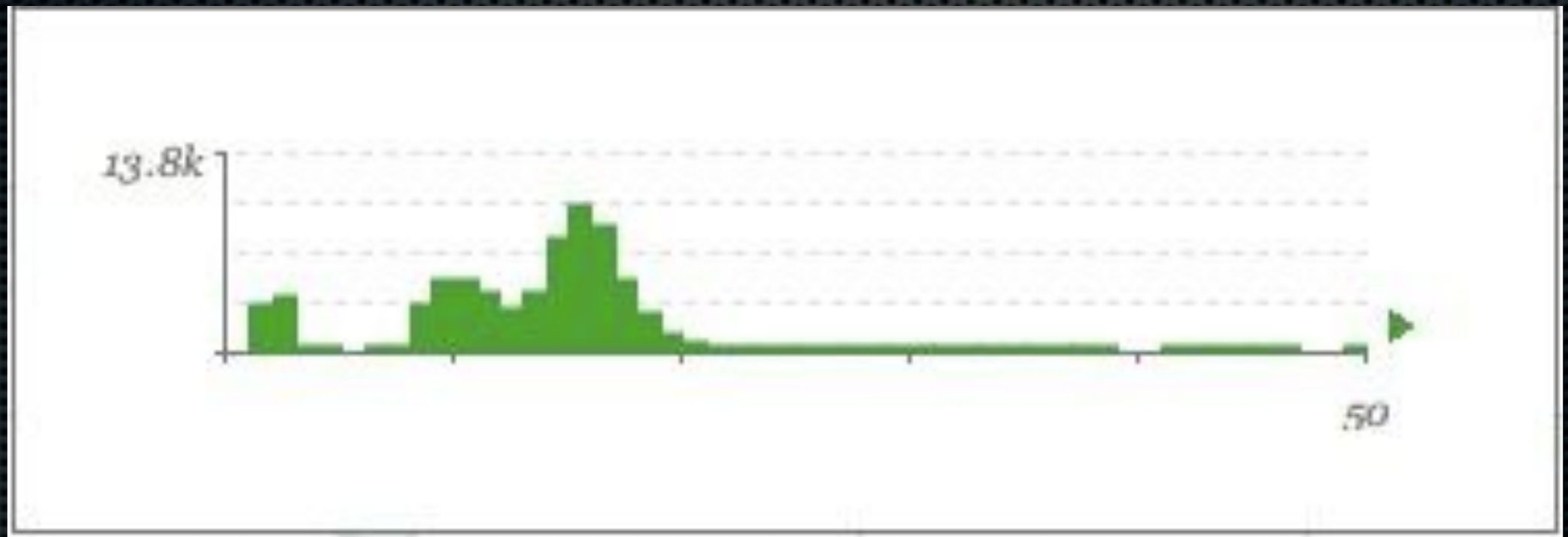
Still… we can do better.

- $\bar{N}$, σ, |N|, q(0,0.5,0.95,0.99,1) is 8 statistical aggregates
- Let's look at API latencies…
  say we do 1000/s,
  that's 60k/minute.
- Over a minute of time, 60k points to 8 represents…
  a lot of information loss.

First 60k/minute, how?

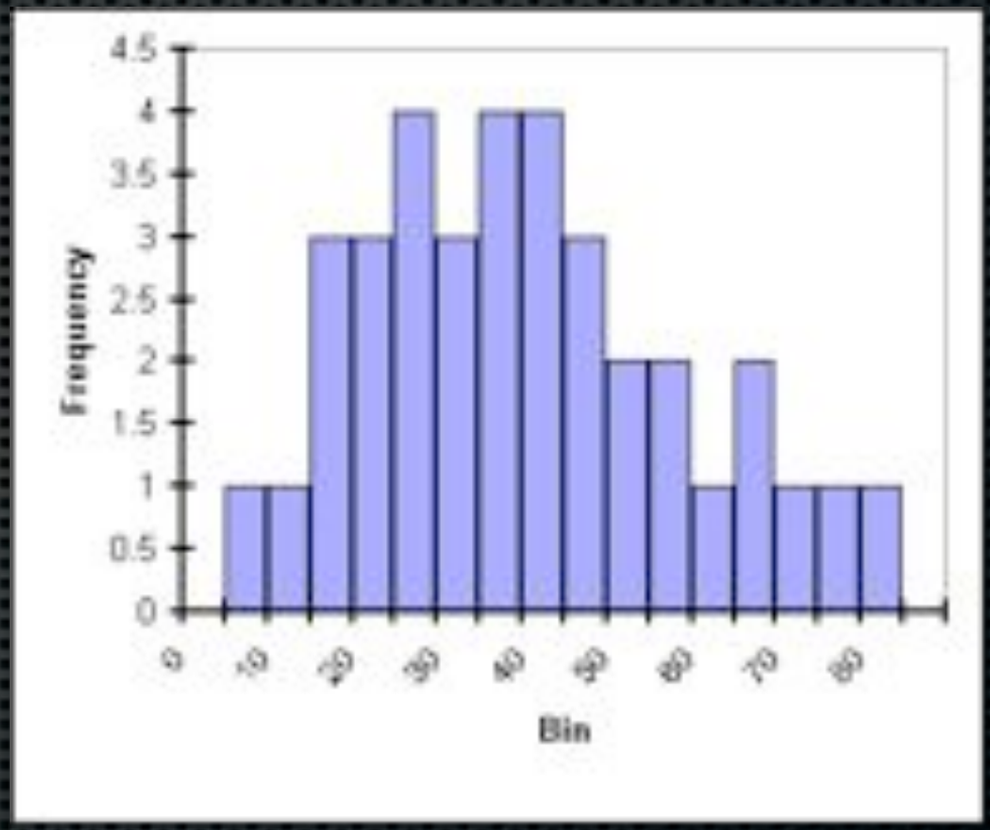
- statsd
- http puts
- logs
- etc.

Histograms



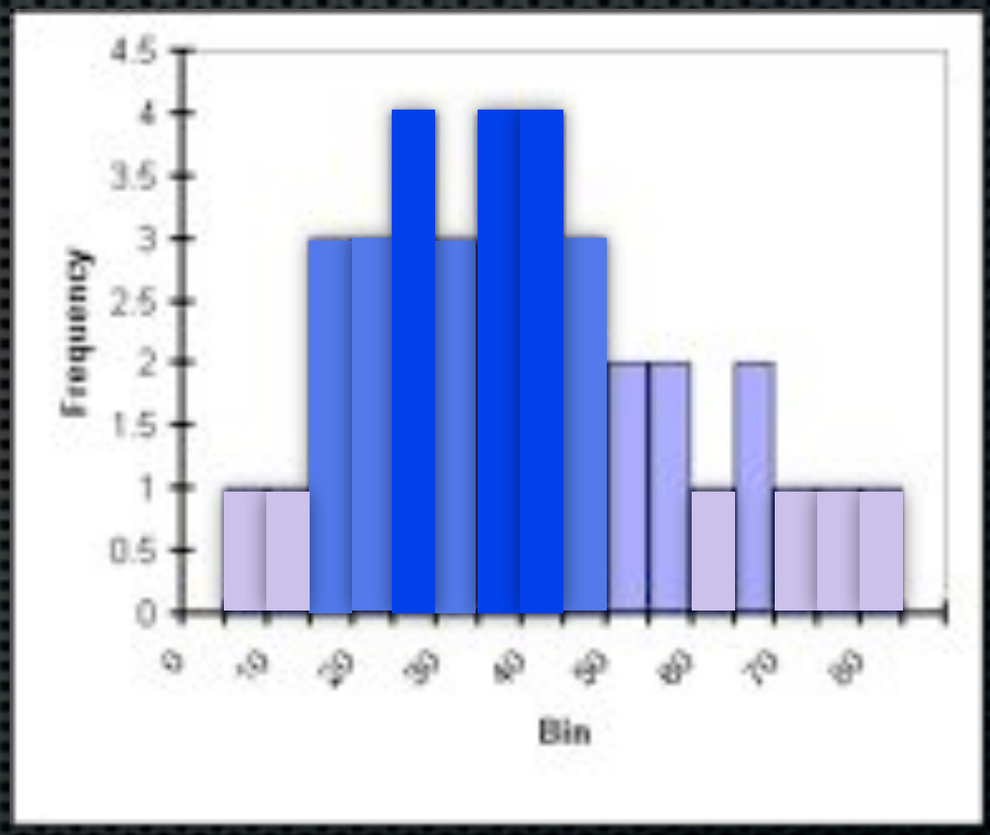
Histograms 101

- This.
- This is a histogram.
- It shows the frequency of values within a population.
- Height represents frequency



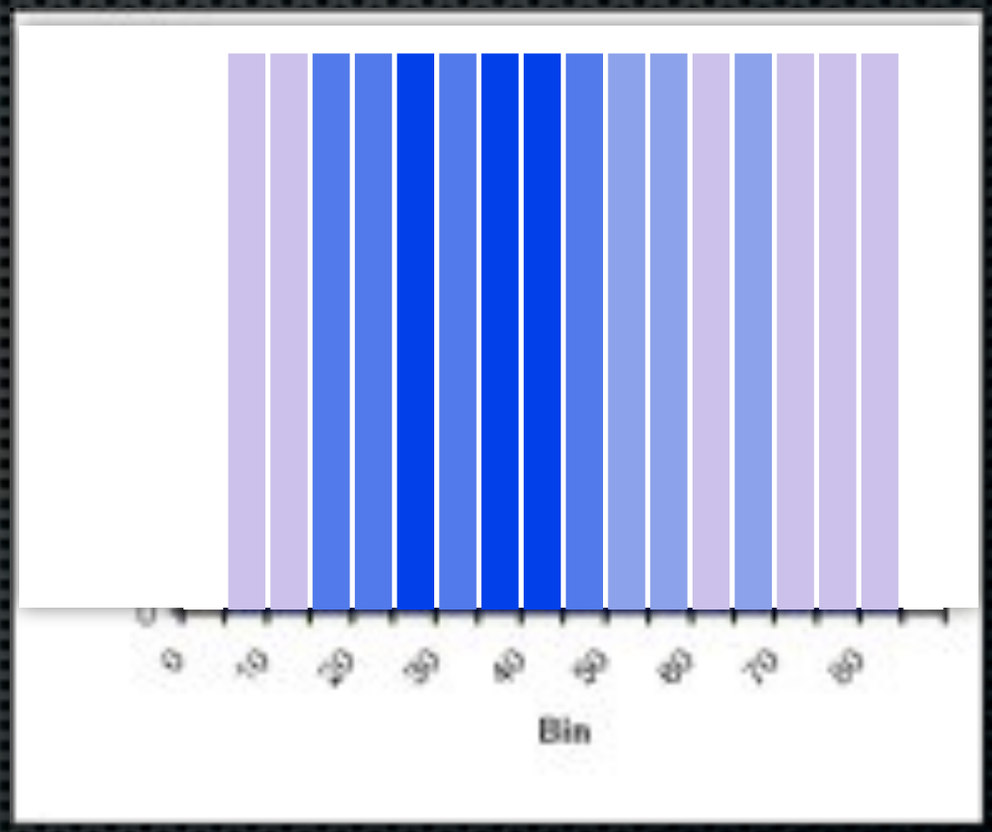
Histograms 101

- This.
- This is a histogram.
- It shows the frequency of values within a population.
- Now, height ***and*** color represents frequency



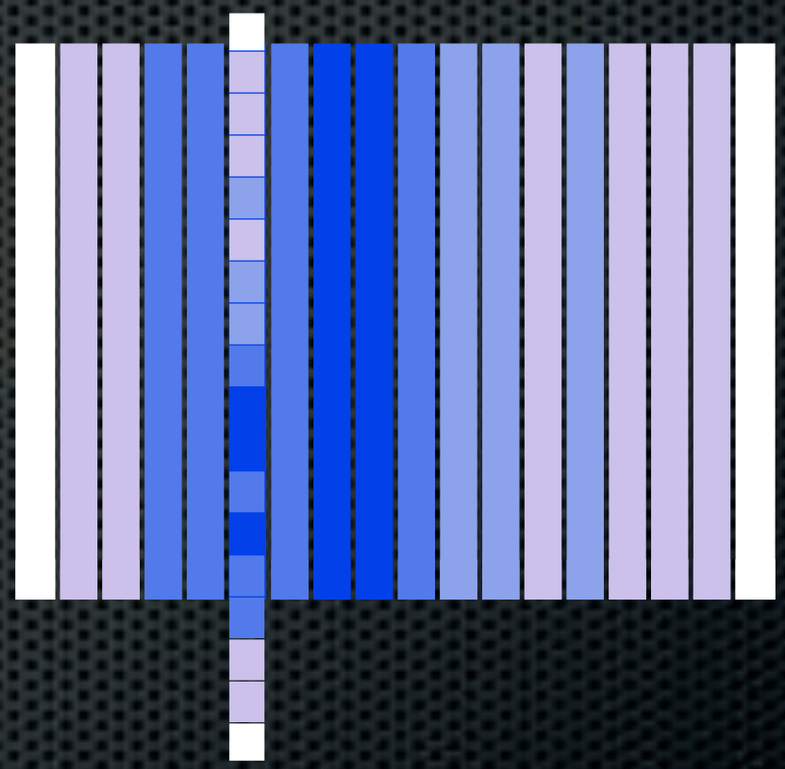
Histograms 101

- This.
- This is a histogram.
- It shows the frequency of values within a population.
- Now, only color represents frequency



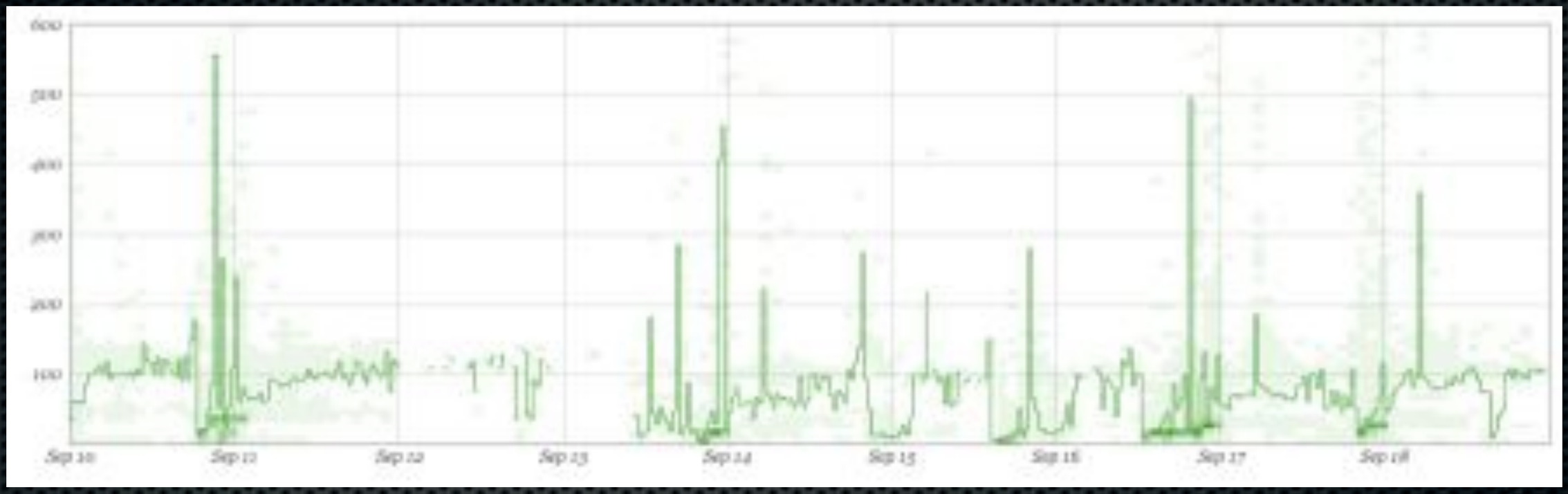
Histograms ➠ time series

- This.
- This is a histogram.
- It shows the frequency of values within a population.
- Now, only color represents frequency

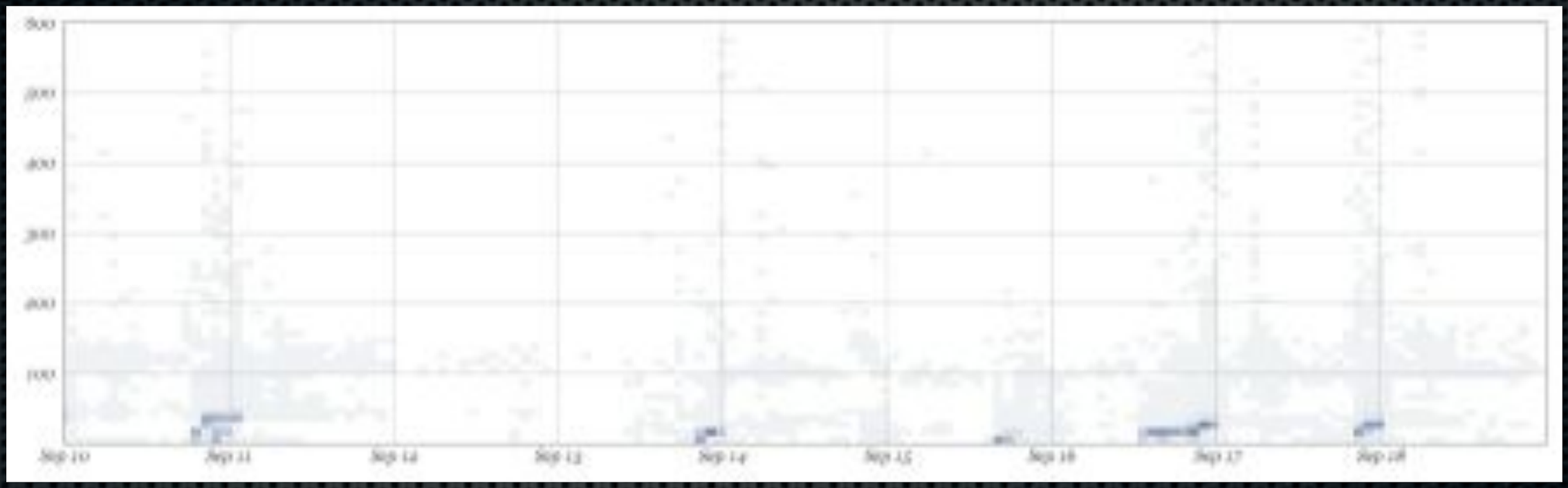


at a single time interval

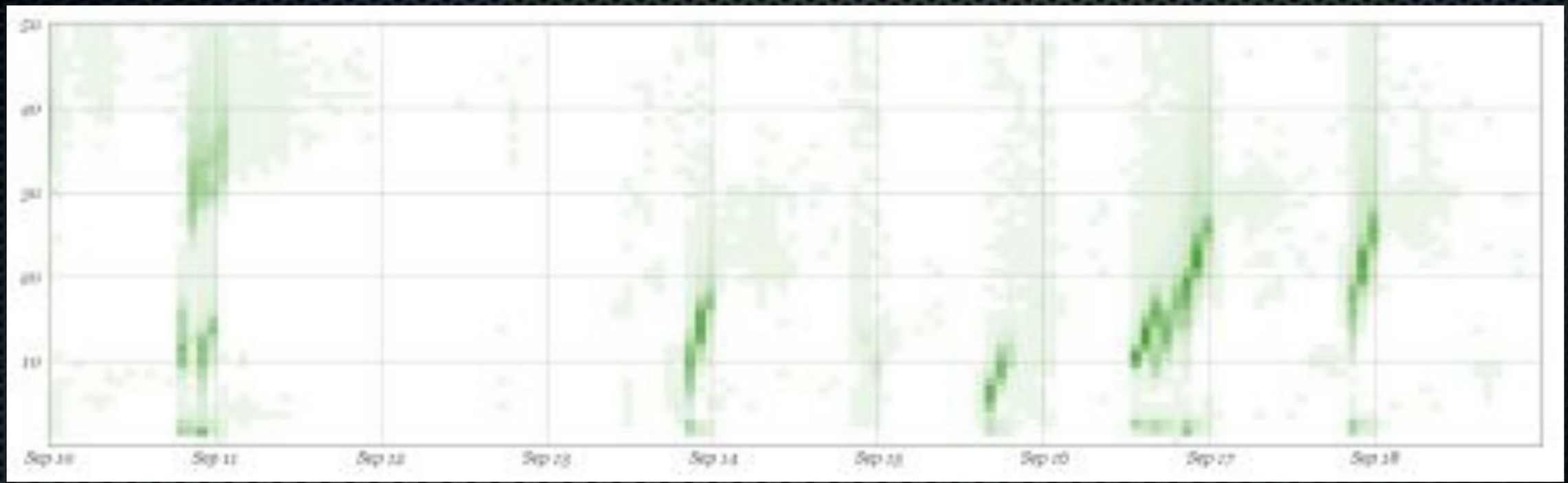
A line graph of data.



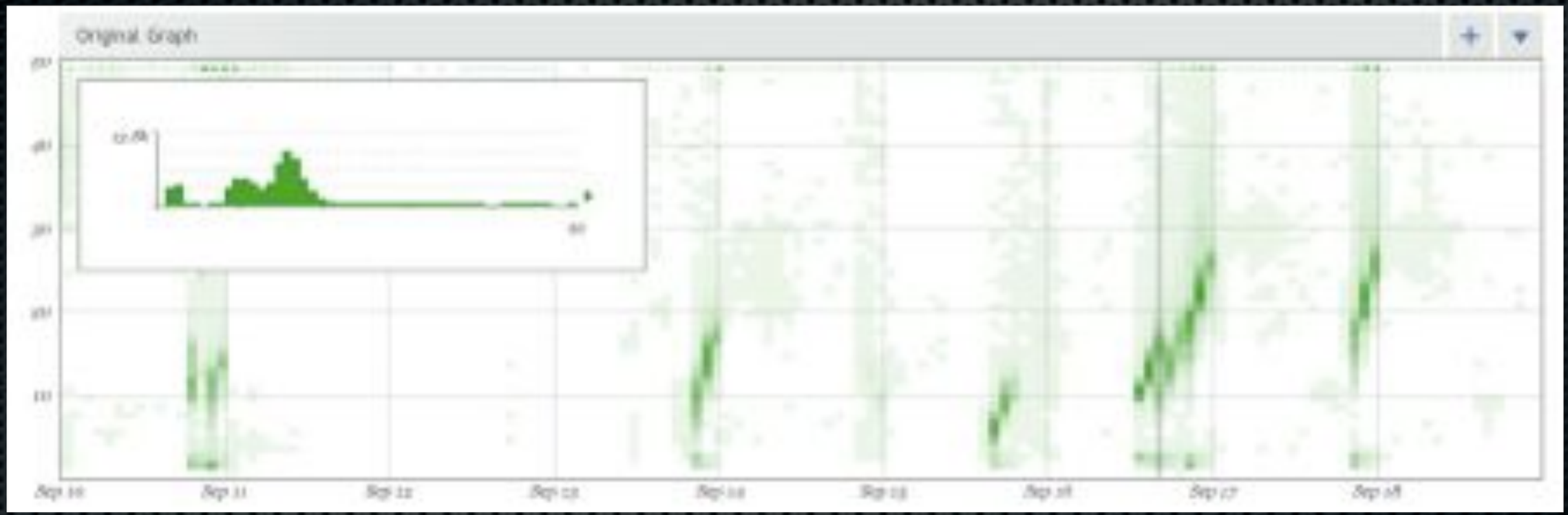
A heatmap of data.



Zoomed in on a heatmap.



Unfolding to a histogram.



Observability

- I don't want to launch into a tutorial on DTrace despite the fact that you can simple spin up an OmniOS AMI in Amazon and have it now.
- Instead let's talk about what shouldn't happen.

The production questions:

- I wonder if that queue is backed up...
- Performance like that should only happen if our binary tree is badly imbalanced (replace with countless other pathologically bad precipitates of failure); I wonder if it is...
- It's almost like some requests are super slow; I wonder if they are.
- STOP WONDERING.

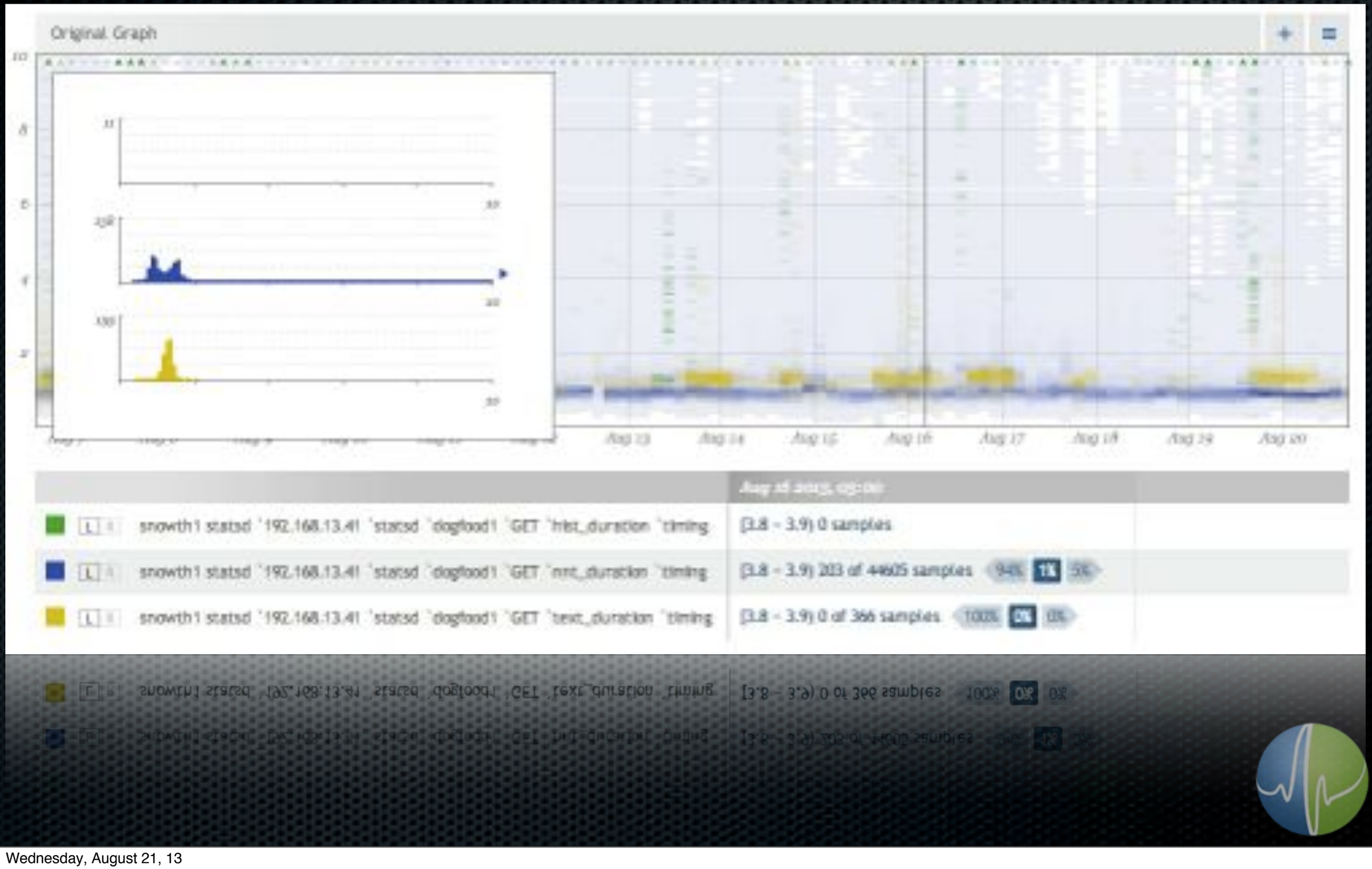
Instrument your software

- Instrument your software and systems and stop the wonder
- Do it for the kids
- This is simple with DTrace & a bit more work otherwise
- Avoiding work is not an excuse for ignorance

A tour through our Sauna

- We have this software that stores data... happens to store all data visualized in Circonus.
- We have to get data into the system.
- We have to get data out of the system.
- I don't wonder... here's why.


538b91ab4b4df70a6f00026e9498dd262f44f555
Original Graph
May 27
May 30
Jun 2
Jun 5
Jun 8
Jun 11
Jun 14
Jun 17
Jun 20


Original Graph
snowth1 statsd '192.168.13.41' 'statsd' 'dogfood1' 'GET' 'hist_duration' 'timing
(3.8 – 3.9) 0 samples
snowth1 statsd '192.168.13.41' 'statsd' 'dogfood1' 'GET' 'mrt_duration' 'timing
(3.8 – 3.9) 203 of 44605 samples
94%
1%
5%
snowth1 statsd '192.168.13.41' 'statsd' 'dogfood1' 'GET' 'text_duration' 'timing
(3.8 – 3.9) 0 of 366 samples
100%
0%
0%

CIRCONUS

PERFORMANCE MONITORING – REVOLUTIONIZED

Summary

Let's review!

Bad habits.

- While monitoring all things is a good approach,
- alerting on things that do not have specific remediation requirements is horribly damaging.

Data tenet.

- Do not collect data twice.
- That which you collect for visualization should be the same data on which you alert.

Alerting tenet.

- A ruleset against metrics in the system should **never** produce an alert without documetation:
  - the failure condition in plain ~~English~~ 中文,
  - the business impact of the failure condition,
  - a concise and repeatable remediation procedure,
  - an escalation path up the chain.

Alerting post mortems

- Try this out:
  - for each alert, run a post mortem exercise
    - understand why it alerted, what was done to fix
    - rehash who the stakeholders are
      have them in the meeting
    - have the stakeholder speak to the business impact

Thank you!