How Twitter Monitors Millions of Time-series

Yann Ramin
Observability at Twitter
Strata Santa Clara - 2014
@theatrus
yann@twitter.com

Monitoring for all of Twitter Services and Infrastructure

Concerns

Time series data

Generating, Collection, Storing, Querying

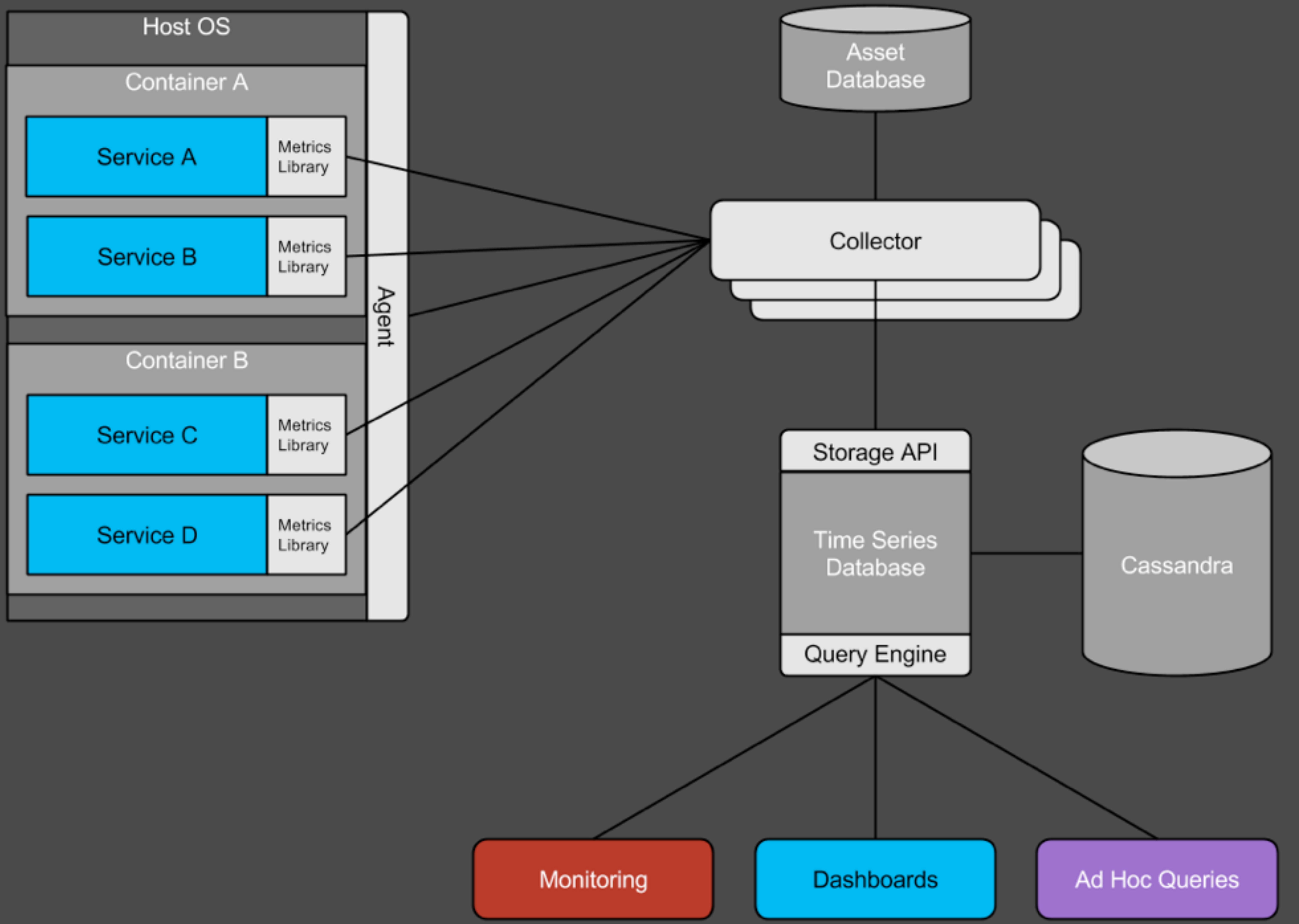
Alerting

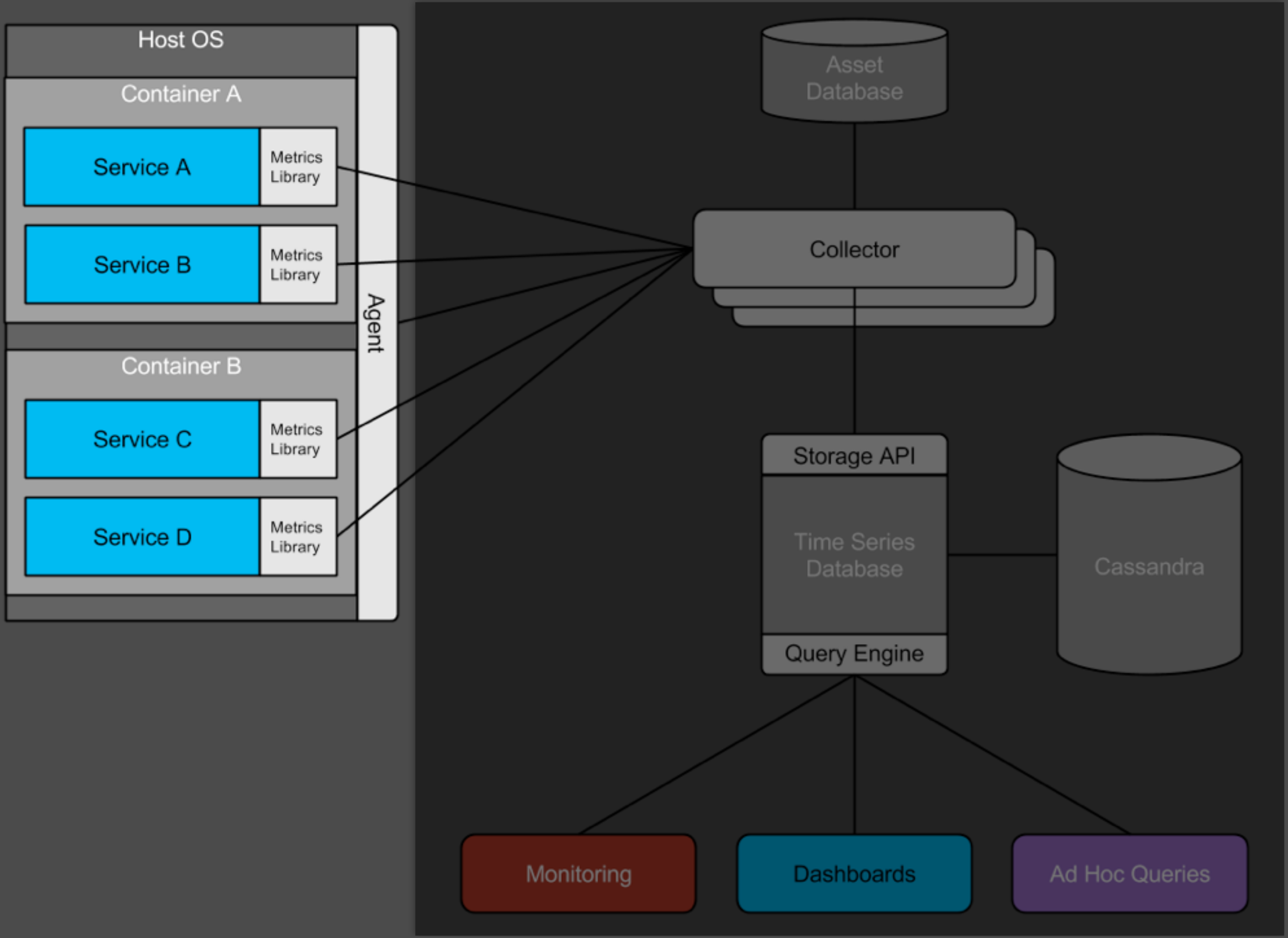
For when you're not watching

Tracing

Distributed systems call tracing

Time series data


Host OS
Container A
Service A
Metrics Library
Service B
Metrics Library
Container B
Service C
Metrics Library
Service D
Metrics Library
Agent
Asset Database
Collector
Storage API
Time Series Database
Query Engine
Cassandra
Monitoring
Dashboards
Ad Hoc Queries


Host OS
Container A
Service A
Metrics Library
Service B
Metrics Library
Container B
Service C
Metrics Library
Service D
Metrics Library
Agent
Asset Database
Collector
Storage API
Time Series Database
Query Engine
Cassandra
Monitoring
Dashboards
Ad Hoc Queries

Data from services

Not just hosts

Contrast:
The “Nagios model”

The website is slow

"Nagios says it can't connect to my webserver"

Why?

```
ssh me@host uptime
ssh me@host top
ssh me@host tail /var/log
```

Now do that for more n > 5 servers

Logs are unstructured

“Log parsing” is a stop-gap

Why deploy log parsing rules with applications?

Move beyond logging

structured statistics

Provide rich and detailed instrumentation

Make it cheap

and easy

First tier aggregations and sampling are in the application

Incrementing atomic counter = cheap
Writing to disk, sending packet, etc = expensive

Lets look at Finagle-based services

http://twitter.github.io/finagle/

Lots of great default instrumentation

For network, JVM, etc

Easy to add more

```
case class StatsFilter(
  name: String,
  statsReceiver: StatsReceiver = NullStatsReceiver
) extends SimpleFilter[Things, Unit] {

  private[this] val stats = statsReceiver.scope(name)
  private[this] val all = stats.counter("all")

  def apply(set: Things, service: Service[Things, Unit]):
Future[Unit] = {
    all.incr(set.length)
    stats.counter(set.service).incr(set.metrics.length)
    service(set)
  }
}
```

```
case class StatsFilter(
  name: String,
  statsReceiver: StatsReceiver
) extends SimpleFilter[Things, Unit] {

                = statsReceiver.scope(name)
                stats.counter("all")

  def apply(set: Things, service: Service[Things, Unit]):
Future[Unit] = {
    all.incr(set.length)
    stats.counter(set.service).incr(set.metrics.length)
    serv
  }
}
```

Get a StatsReceiver

Make a scoped receiver

Create a counter named all

Increment the counter

Get a counter named by variable, increment by length

Easy to get out

http://server:port/
admin/metrics.json

```
{...
"srv/http/request_latency_ms.avg": 45,
"srv/http/request_latency_ms.count": 181094,
"srv/http/request_latency_ms.max": 5333,
"srv/http/request_latency_ms.min": 0,
"srv/http/request_latency_ms.p50": 37,
"srv/http/request_latency_ms.p90": 72,
"srv/http/request_latency_ms.p95": 157,
"srv/http/request_latency_ms.p99": 308,
"srv/http/request_latency_ms.p9990": 820,
"srv/http/request_latency_ms.p9999": 820,
"srv/http/request_latency_ms.sum": 8149509,
"srv/http/requests": 18109445,
```

Great support for approximate histograms

com.twitter.common.stats.ApproximateHistogram

used as stats.stat(“timing”).add(datapoint)

Also, counters & gauges

Twitter-Server

A simple way to make a Finagle server

Sets things up the *right way*

https://github.com/twitter/twitter-server

What about everything else?

Very simple HTTP+JSON protocols means this is easy to add to other *persistent* servers

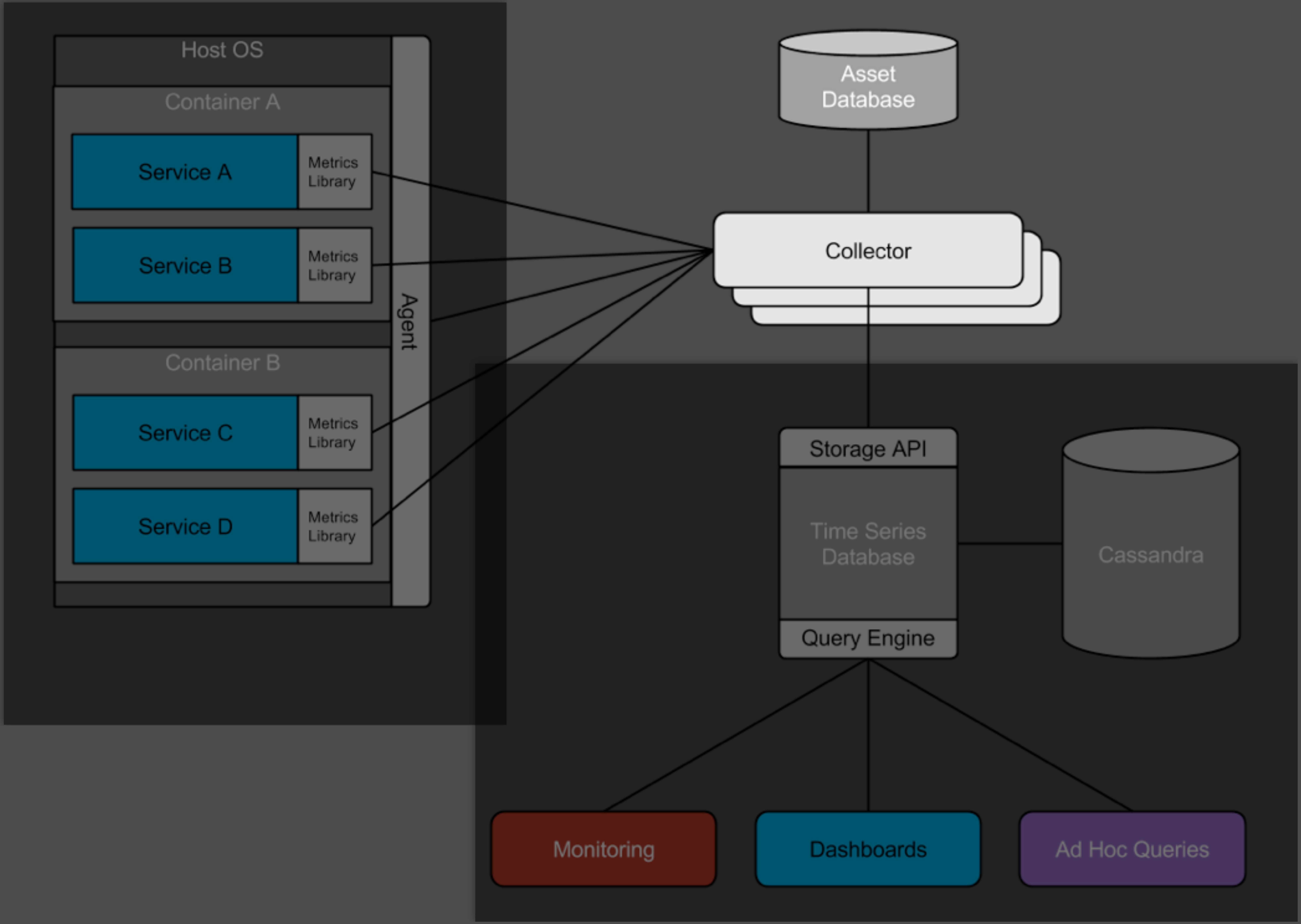
We support ephemeral tasks

Rolls up into a persistent server

Now we’ve replaced ssh with curl

and this is where Observability comes in


Collection



Host OS
Container A
Service A
Metrics Library
Service B
Metrics Library
Container B
Service C
Metrics Library
Service D
Metrics Library
Agent
Asset Database
Collector
Storage API
Time Series Database
Query Engine
Cassandra
Monitoring
Dashboards
Ad Hoc Queries

Distributed Scala service

Find endpoints:

Zookeeper
Asset database
other sources

Fetch/sample data

HTTP GET (*via Finagle*)

Filter, cleanup, etc

Hygiene for incoming data

Route to storage layers!

Time series database, memory pools, queues and HDFS aggregators

Metrics are *added* by default

Need instrumentation? Just add it!
Shows up “instantly” in the system

This is good

Easy to use
No management overhead

"Can you add a rrd-file for us?"

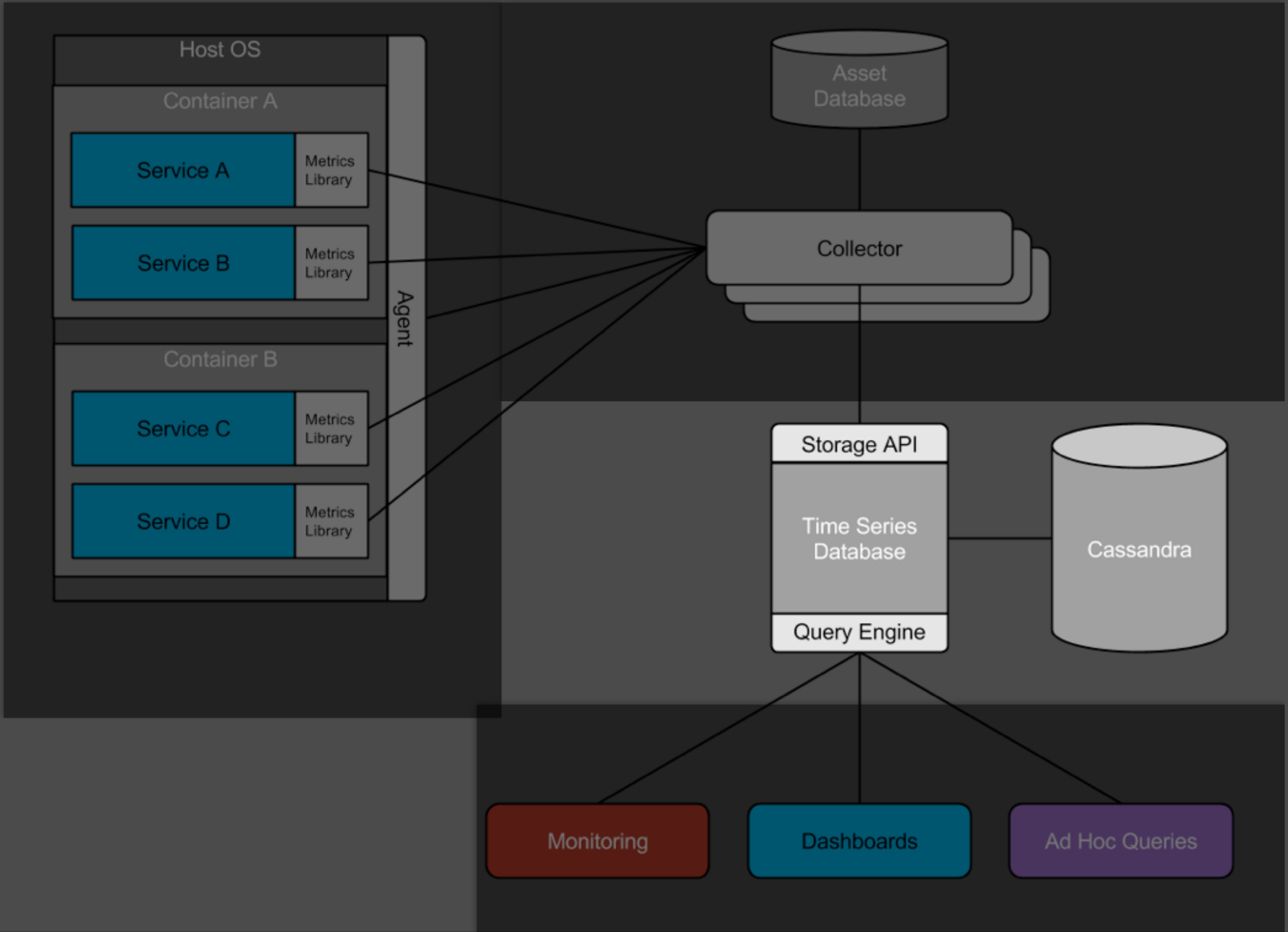
This is bad

"Metric name on .toString"
[I@579b7698

Remove barriers
Be defensive

Pick both


Time series storage


Host OS
Container A
Service A
Metrics Library
Service B
Metrics Library
Container B
Service C
Metrics Library
Service D
Metrics Library
Agent
Asset Database
Collector
Storage API
Time Series Database
Query Engine
Cassandra
Monitoring
Dashboards
Ad Hoc Queries

Distributed Scala front-end service

Databases, caches, data conversion, querying, etc.

220 million time series

Updated every minute

When this talk was proposed: 160 million time series

Cassandra

For real time storage

(Now replaced with an internal database)

Similar enough to Cassandra

Uses KV storage

For the most part

Multiple clusters per DC

For different access patterns

We namespace metrics

Service = group
Source = where
Metric = what

Row key:
(service, source, metric)

Columns:
timestamp = value

Range scan for time series

Tweaks: Optimizations for time series

We never modify old data
We time-bound old data writes

Informed heuristics to reduce SSTables scanned

Easy expiry - drop the whole SSTable

Cassandra Counters

Write time aggregations

“Services as a whole”

Why read every “source” all the time?
Write them all into an aggregate

Don’t scale with cluster size

Limited aggregations

Sum, Count

Non-idempotent writes

Bad failure modes

Over counting? Undercounting? Who knows!

Friends don’t let friends use counters

http://aphyr.com/posts/294-call-me-maybe-cassandra

Expanding storage tiers

Memcache
HDFS Logs
On-demand high resolution samplers


Name indexing

What metrics exist?
What instances? Hosts?
Services?

Used in language tools (globs, etc) and discovery tools (here is what you have)

Index is temporal

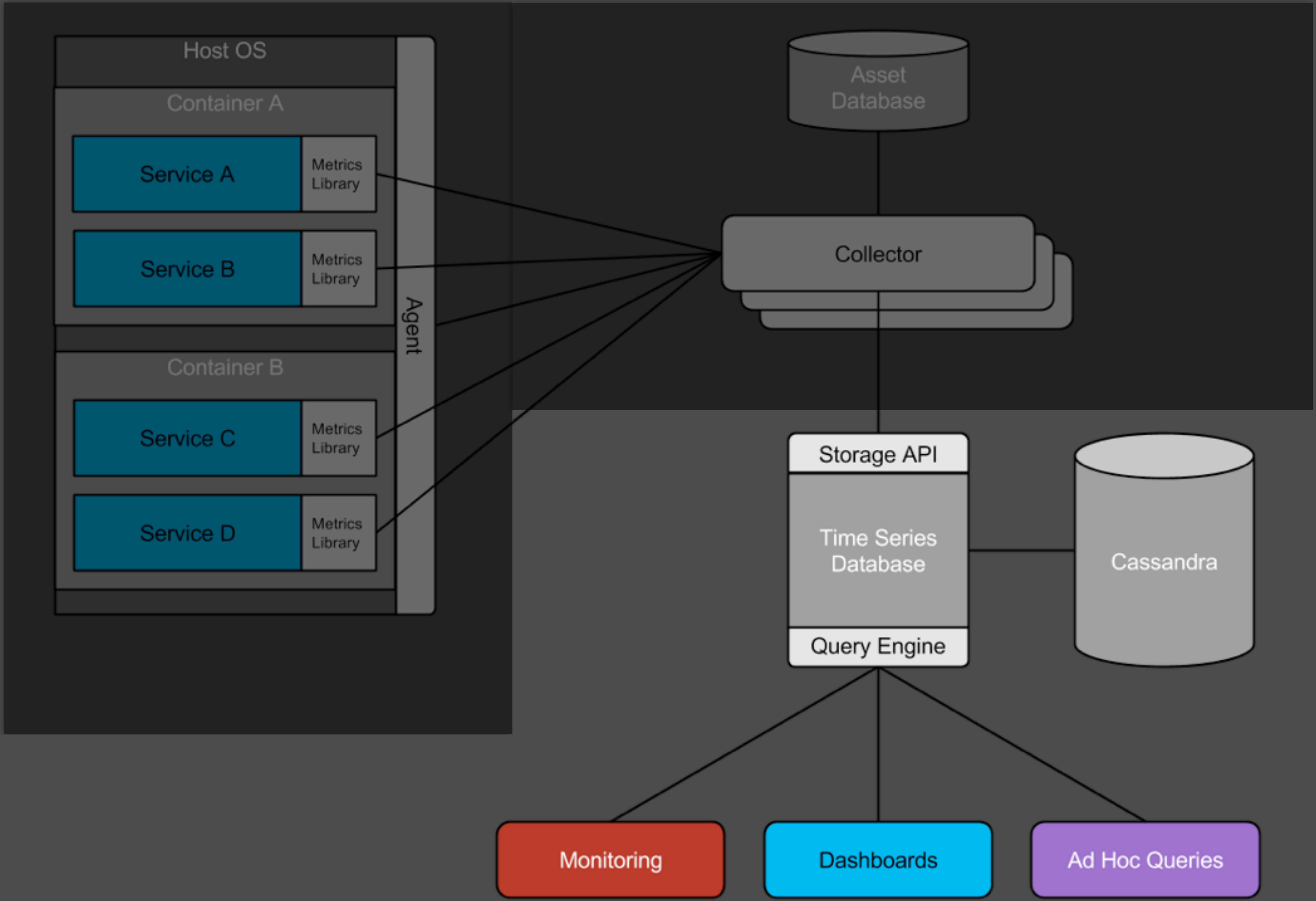
“All metrics matching `http/*`, from Oct 1-10”

Maintained as a log of operations on a set t = 0, add metric r

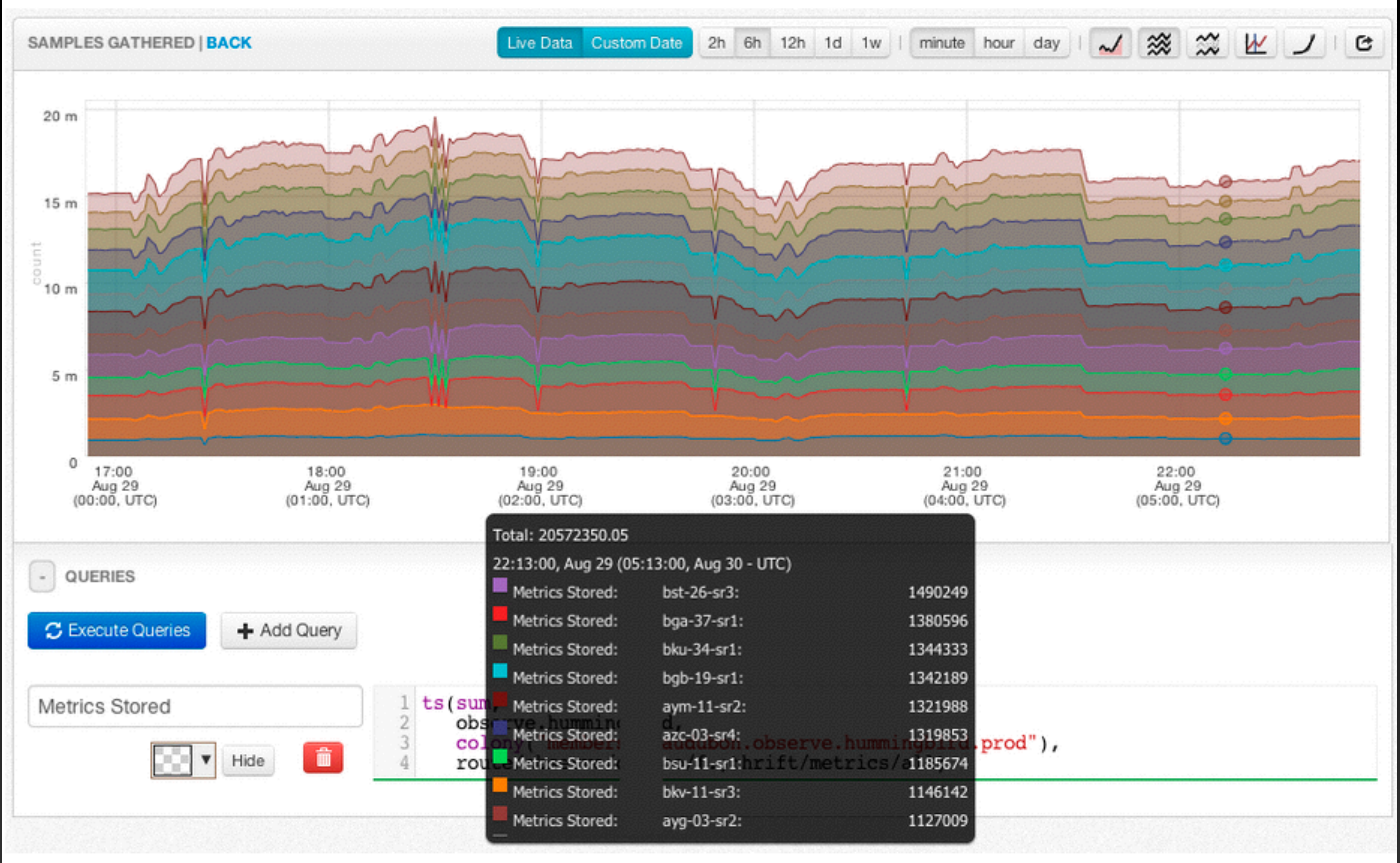
t = 2, remove metric q

Snapshot to avoid long scans

Getting data


Host OS
Container A
Service A
Metrics Library
Service B
Metrics Library
Container B
Service C
Metrics Library
Service D
Metrics Library
Agent
Asset Database
Collector
Storage API
Time Series Database
Query Engine
Cassandra
Monitoring
Dashboards
Ad Hoc Queries

Ad-hoc queries

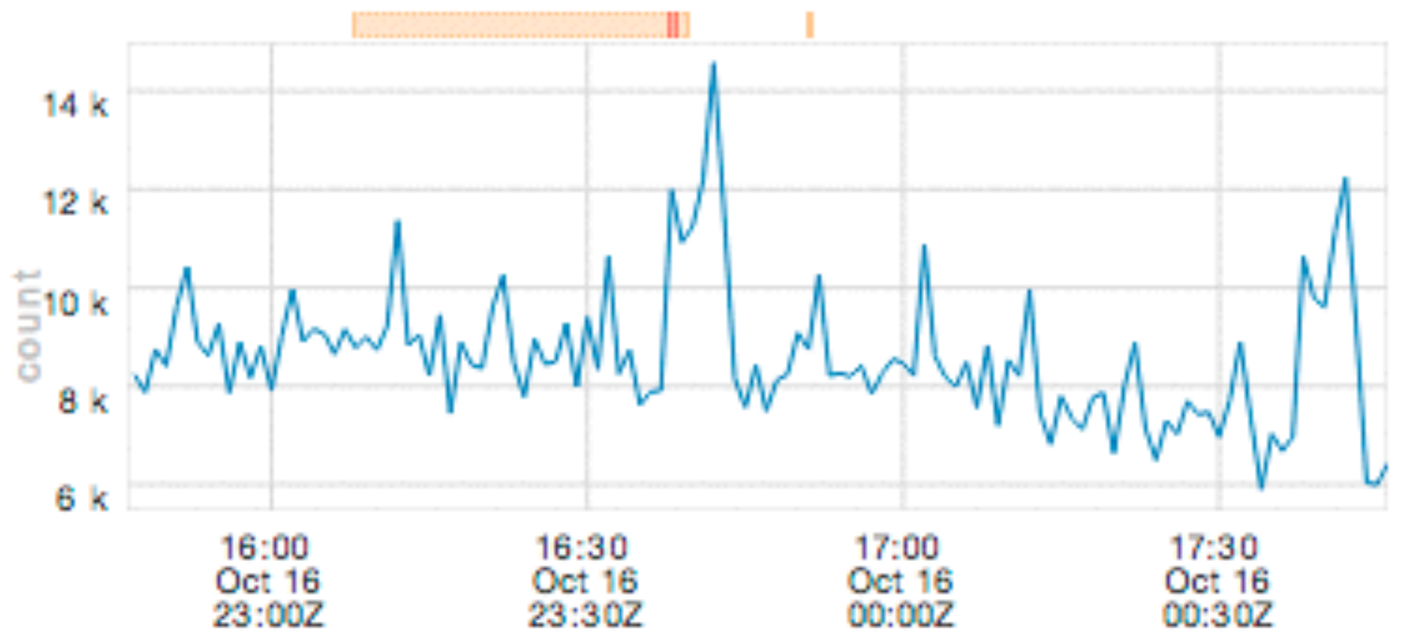


SAMPLES GATHERED | BACK
Live Data
Custom Date
2h
6h
12h
1d
1w
minute
hour
day
20 m
15 m
10 m
5 m
0
count
17:00 Aug 29 (00:00, UTC)
18:00 Aug 29 (01:00, UTC)
19:00 Aug 29 (02:00, UTC)
20:00 Aug 29 (03:00, UTC)
21:00 Aug 29 (04:00, UTC)
22:00 Aug 29 (05:00, UTC)
Total: 20572350.05
22:13:00, Aug 29 (05:13:00, Aug 30 - UTC)
Metrics Stored: bst-26-sr3: 1490249
Metrics Stored: bga-37-sr1: 1380596
Metrics Stored: bku-34-sr1: 1344333
Metrics Stored: bgb-19-sr1: 1342189
Metrics Stored: aym-11-sr2: 1321988
Metrics Stored: azc-03-sr4: 1319853
Metrics Stored: bsu-11-sr1: 1185674
Metrics Stored: bkv-11-sr3: 1146142
Metrics Stored: ayg-03-sr2: 1127009
QUERIES
Execute Queries
Add Query
Metrics Stored
Hide
ts(sum
.prod"),
hrift/metrics/a

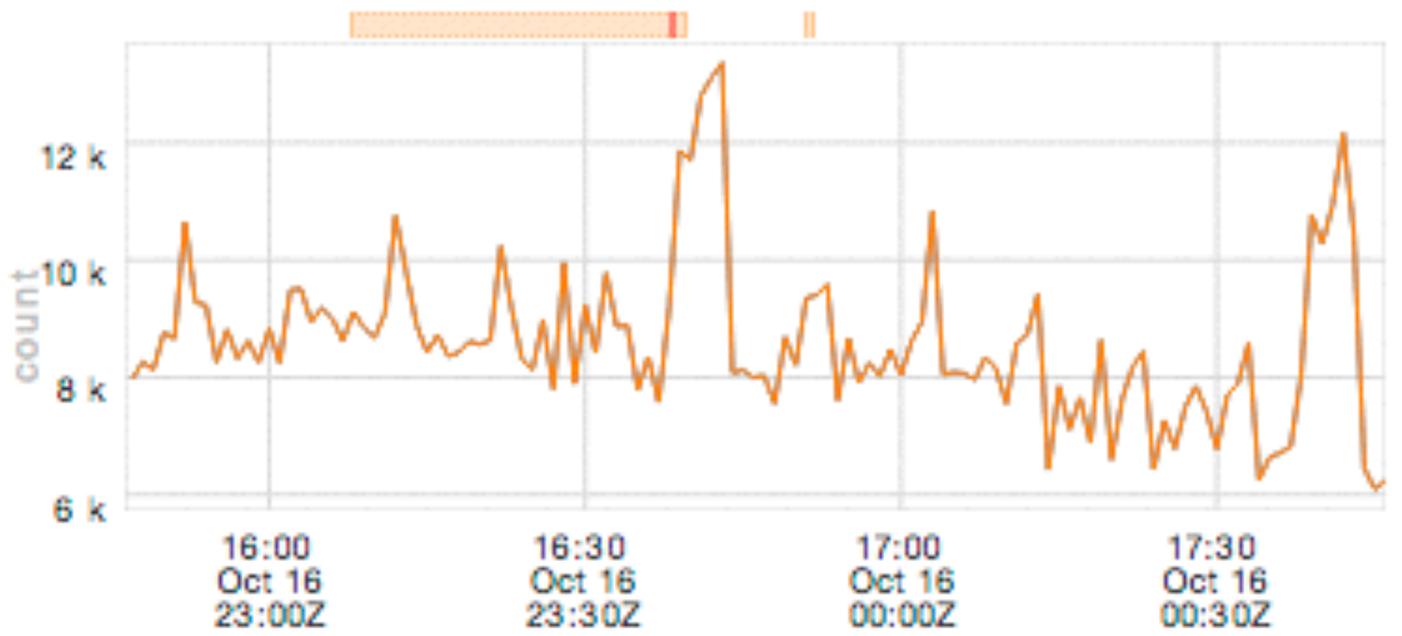
Dashboards

READ: Connections

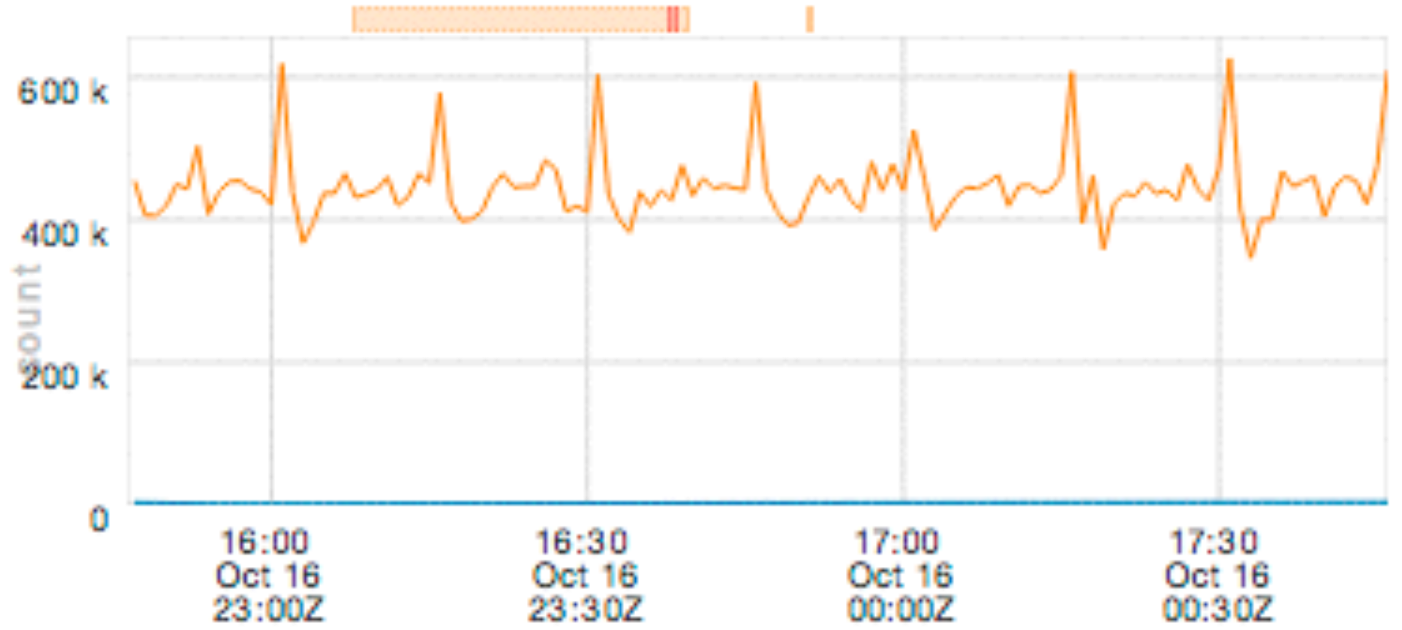
HTTP connections



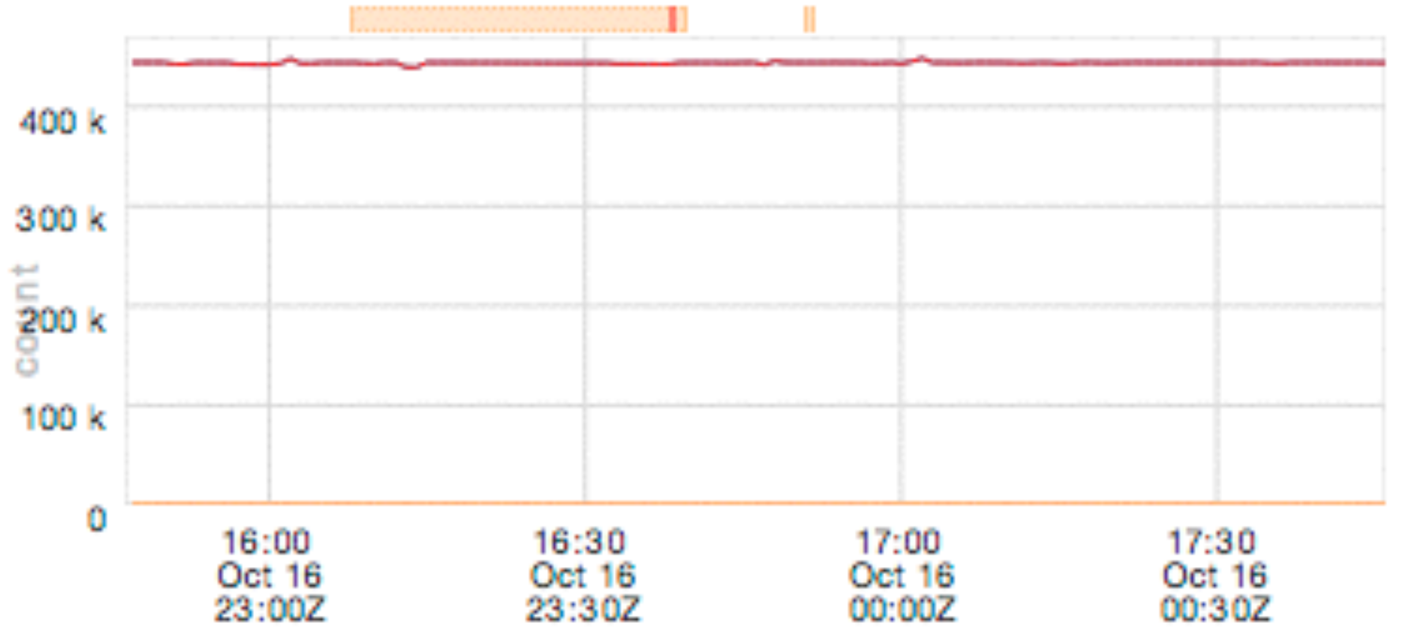
HTTP requests



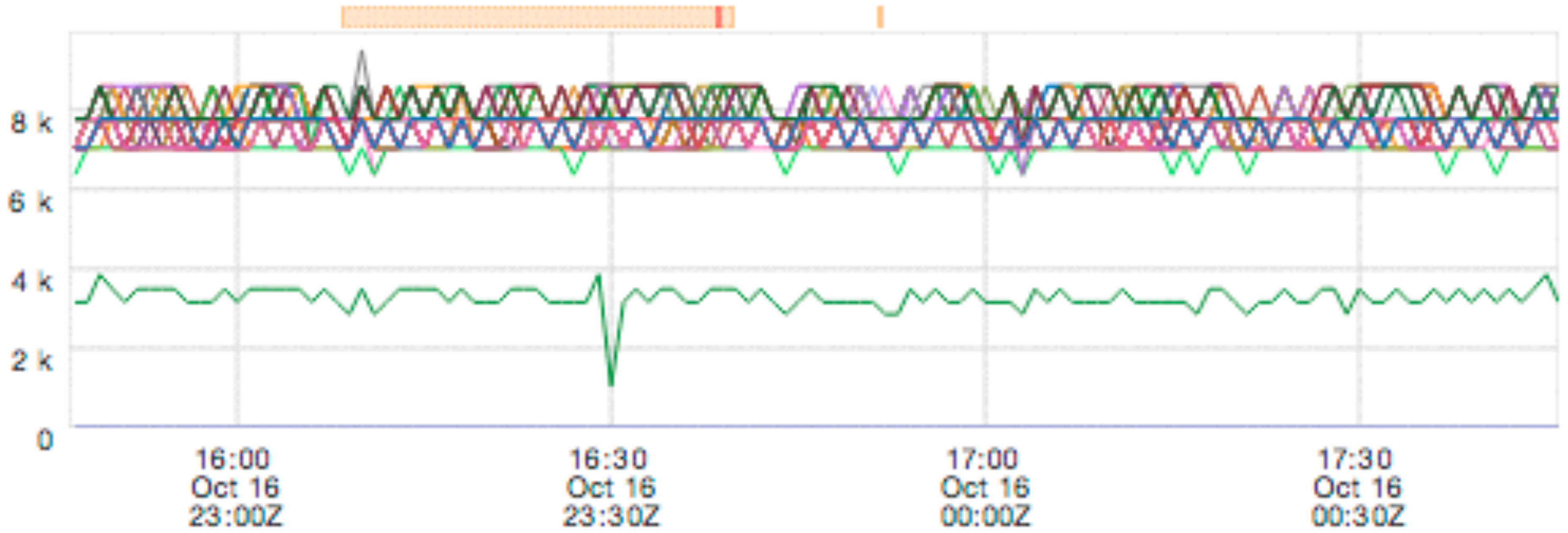
Thrift connections



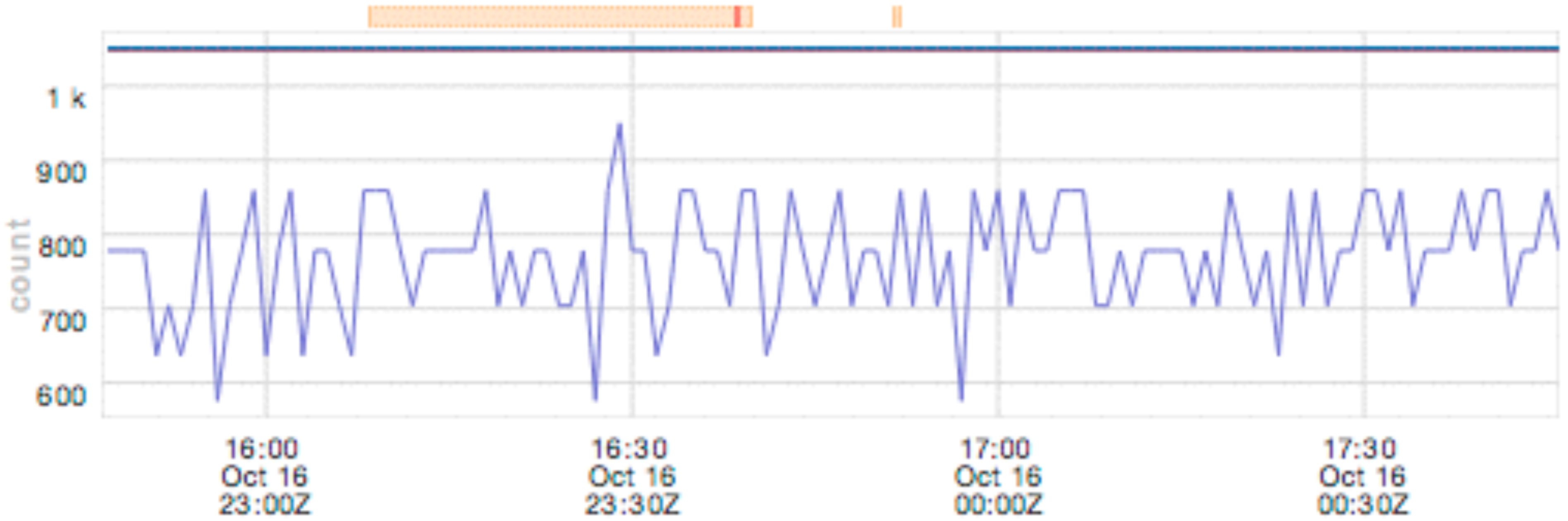
Thrift requests

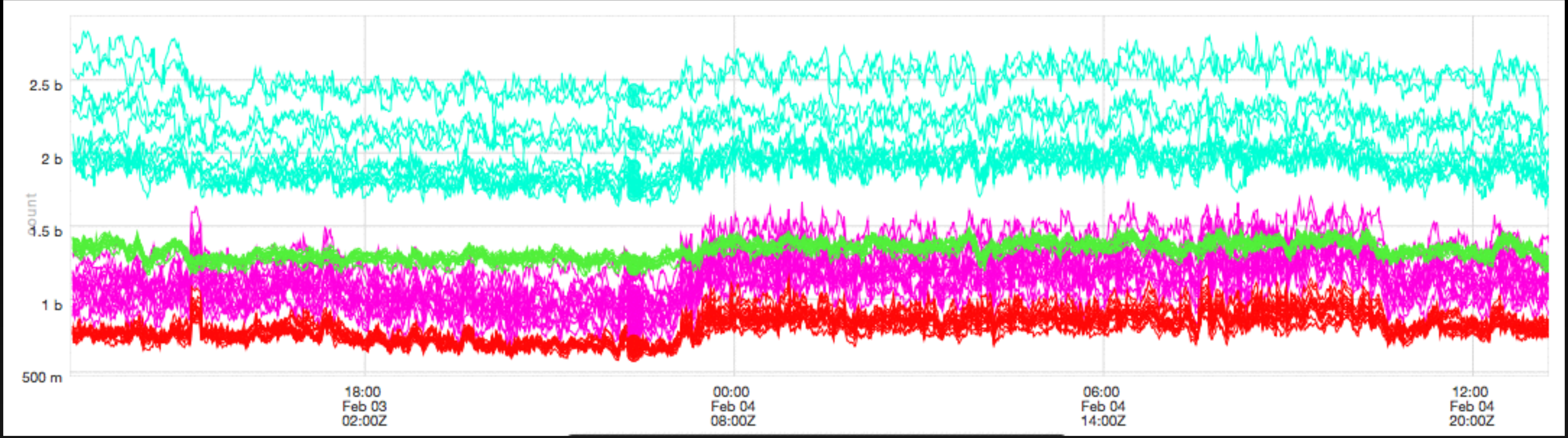


Cassandra Write Queue High Watermark



Cassandra Write Queue Unavailable Permits





QUERIES

Execute Queries | + Add Query

rx nh — Hide | Duplicate

```
movingavg(10, rate(ts(sum, system.network,  members(nuthatch), if/eth0/rx_bytes)))
```

tx index — Hide | Duplicate

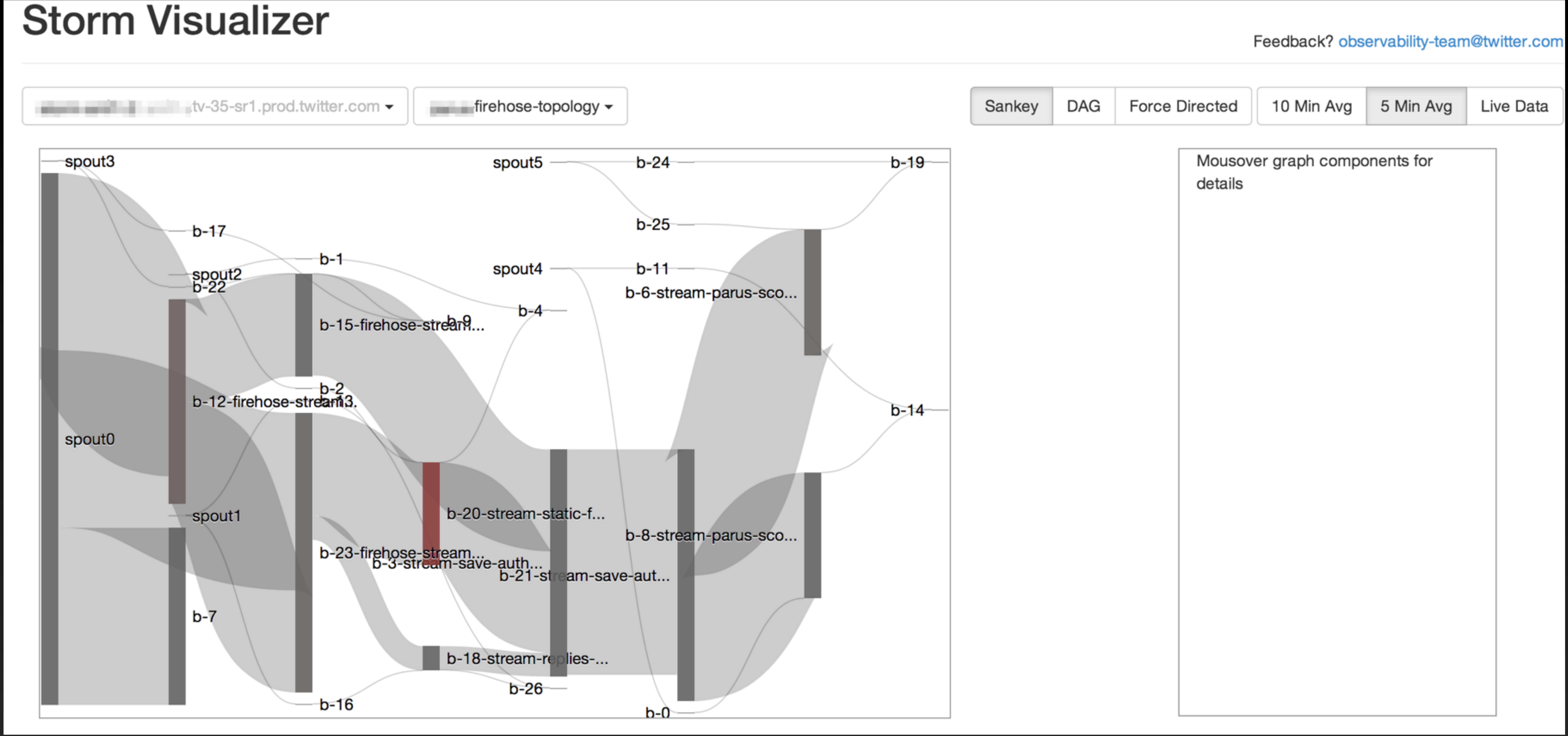
```
movingavg(10, rate(ts(sum, system.network, members(db.index), if/eth0/tx_bytes)))
```

rx index — Hide | Duplicate

```
movingavg(10, rate(ts(sum, system.network, members(db.index), if/eth0/rx_bytes)))
```

tx nh — Hide | Duplicate

```
movingavg(10, rate(ts(sum, system.network,  members(nuthatch), if/eth0/tx_bytes)))
```



Specialized Visualizations: Storm

Everything is built on our query language

CQL

Not the Cassandra one

Functional/declarative language

On-demand

Don’t need to pre-register queries

Aggregate, correlate and explore

Metric Aggregate Functions

Aggregate functions compute the aggregate at each step over a set of timeseries,

Function	Description
`sum( ts )`	sum of all time series datapoints into a new timeseries. V
`avg( ts )`	average of all time series datapoints into a new timeserie
`min( ts )`	minimum datapoint from all timeseries (see also: `percen`
`max( ts )`	maximum datapoint from all timeseries (see also: `perce`
`count( ts )`	count of non-null values.
`percentile( ptile , ts )`	`ptile` percentage of the timeseries set. `ptile`'s valid ra the median value can be returned via `percentile(50,`
`stddev(ts)`	The standard deviation of the set of timeseries. Returns

Timeseries Grouping Functions (groupby)

In combination with *metric aggregate functions*, it is possible to compute aggregates

The principal function is `groupby(dimension, aggregation, input)`

Non-aggregate Functions

Non-aggregate functions operate on each timeseries in a set independently, re

Function	Description
`rate( ts )`	**Positive** rate of change (derivative) with a s Rate can never be negative - it is designed f resets or roll-overs which `derive` would not. smooth windows allow for glitchy counters to
`default( default , ts )`	Replaces `null` values with *`default`* . Note
`or( primary , fallback )`	Returns primary unless it is an empty set of

and many more (cross-DC federation, etc)

Support matchers and drill down from index

i.e., Explore by regex: ***http*latency.p9999***

Get and combine two time series

Ratio of GC activity to requests served

Get and combine two time series

We didn't create a stat :(

Get and combine two time series

But, we can query it!

Get and combine two time series

```
ts(cuckoo, members(role.cuckoo_frontend),
jvm_gc_msec) /
ts(cuckoo, members(role.cuckoo_frontend),
api/query_count)
```

Queries work with "interactive" performance

When something is wrong, you need data *yesterday*

p50 = 2 milliseconds

p9999 = 2 seconds

Support individual time series and aggregates

Common to aggregate 100-10,000 time series

Over a week
Still respond within 5 seconds, cold cache

Aggregate partial caching

Cache this result!

```
max(rate(ts( {10,000 time series match }))
```

Time limiting out-of-order arrivals makes this a safe operation

Caching via Memcache

Time-windowed immutable results
e.g. 1-minute, 5-minute, 30-minute, 3-hour immutable spans

Replacing with an internal time series optimized cache

Read federations

Tiered storage:
High temporal resolution, caches, long retention

Different data centers and storage clusters

Read federations

Decomposes query, runs fragments next to storage

On-demand secondly resolution sampling

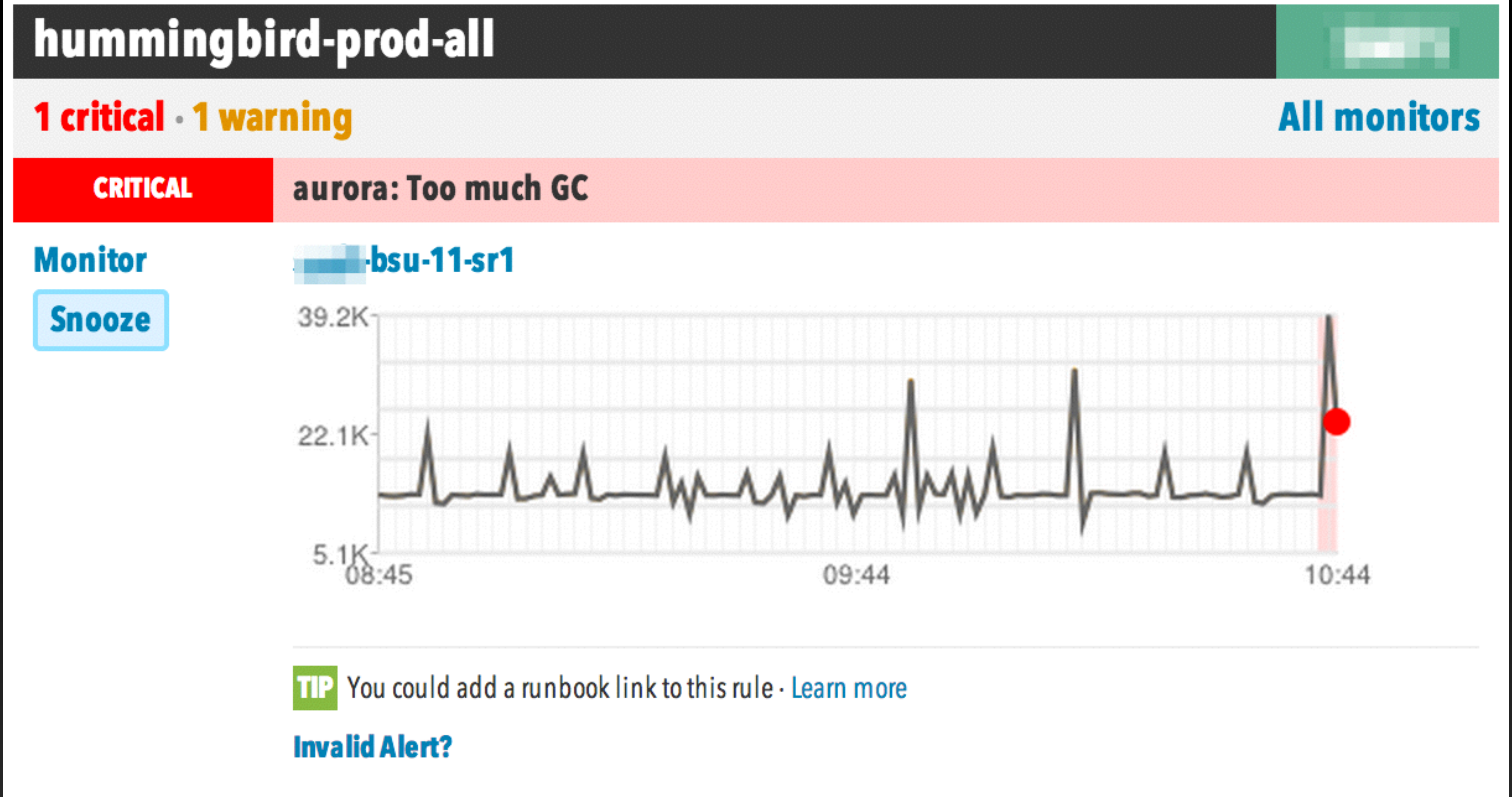
Launch sampler in Apache Mesos
Discovery for read federation is automatic

Query system uses a term rewriter structure

Multi-pass optimizations
Data source lookups
Cache modeling
Costing and very large query avoidance
Inspired by Stratego/XT


Alerting


home
viz
alerts
exceptions
tracing
Access this alert in another zone!
Rules
State History
Alert Definition
cuckoo
Owned by observability-team (edit)
Snooze this alert
Monitor States
222 ok
Showing All
Snoozers
Showing All
Rule
Status
JVM uptime is low
73
GC latency is high
73
Free heap space is low
73
Several nodes are not reporting metrics
1
Read errors are high
1
Read success is low
1



Paging and e-mails

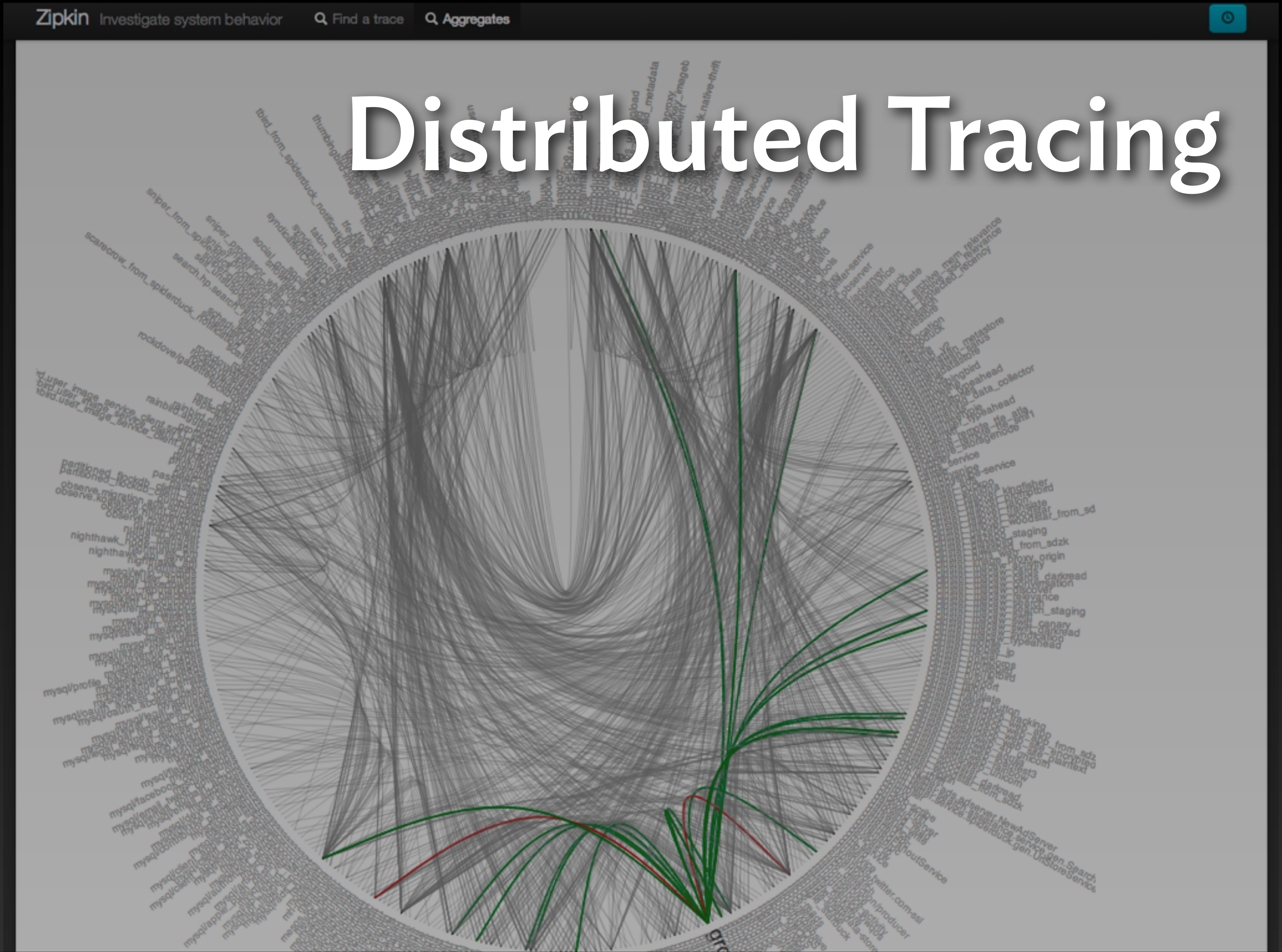
Uses CQL

Adds predicates for conditions
See, unified access is a good thing!

Widespread

Watches all key services at Twitter

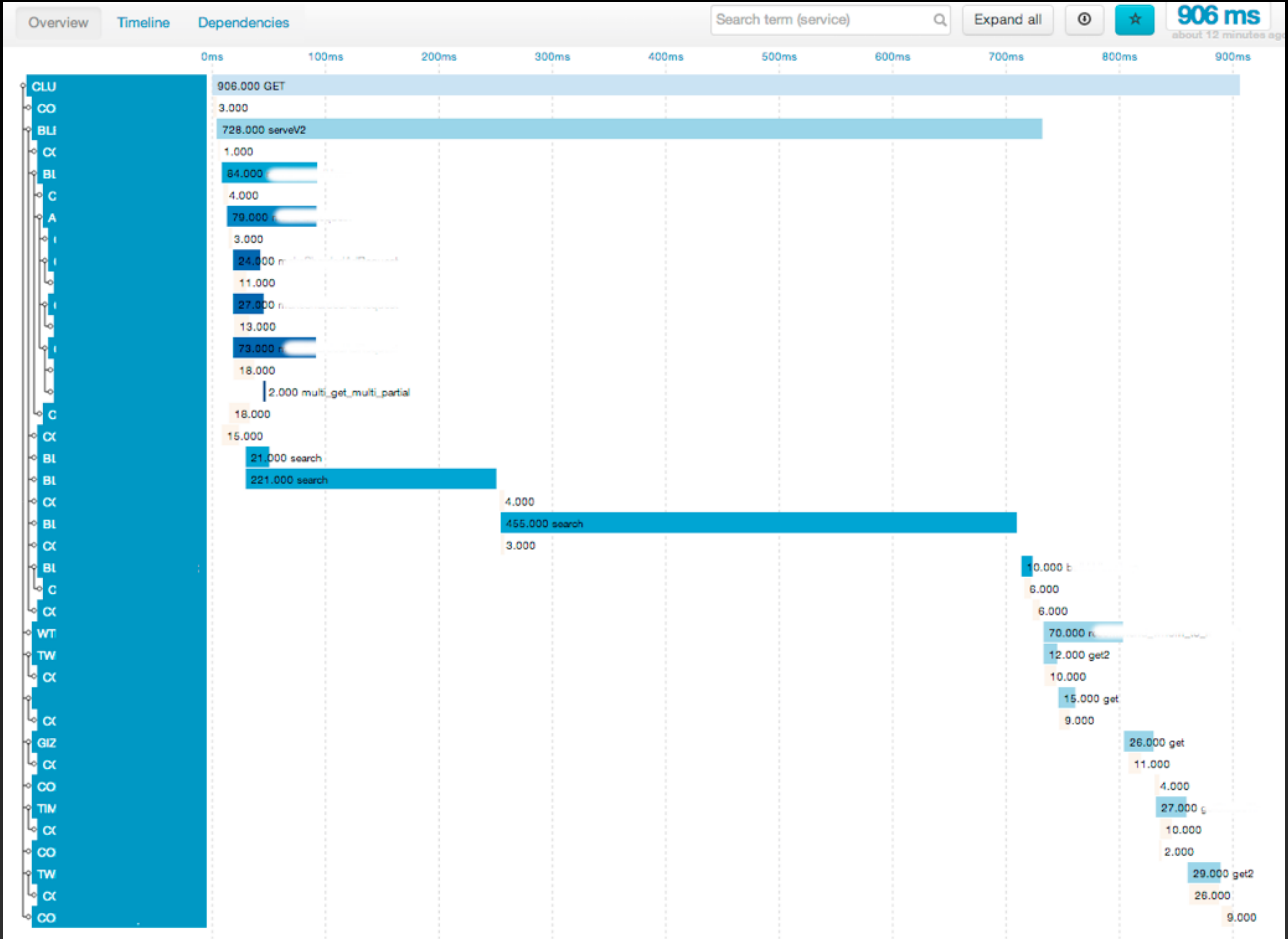
Distributed Tracing



Zipkin

https://github.com/twitter/zipkin

Based on the Dapper paper



Overview
Timeline
Dependencies
Search term (service)
Expand all
906 ms
0ms
100ms
200ms
300ms
400ms
500ms
600ms
700ms
800ms
900ms
906.000 GET
728.000 serveV2
2.000 multi_get_multi_partial
21.000 search
221.000 search
455.000 search
12.000 get2
15.000 get
26.000 get
29.000 get2

Sampled traces of services calling services

Hash of the *trace ID* mapped to sampling ratio

Annotations on traces

Request parameters, internal timing, servers, clients, etc.

Finagle “upgrades” the Thrift protocol

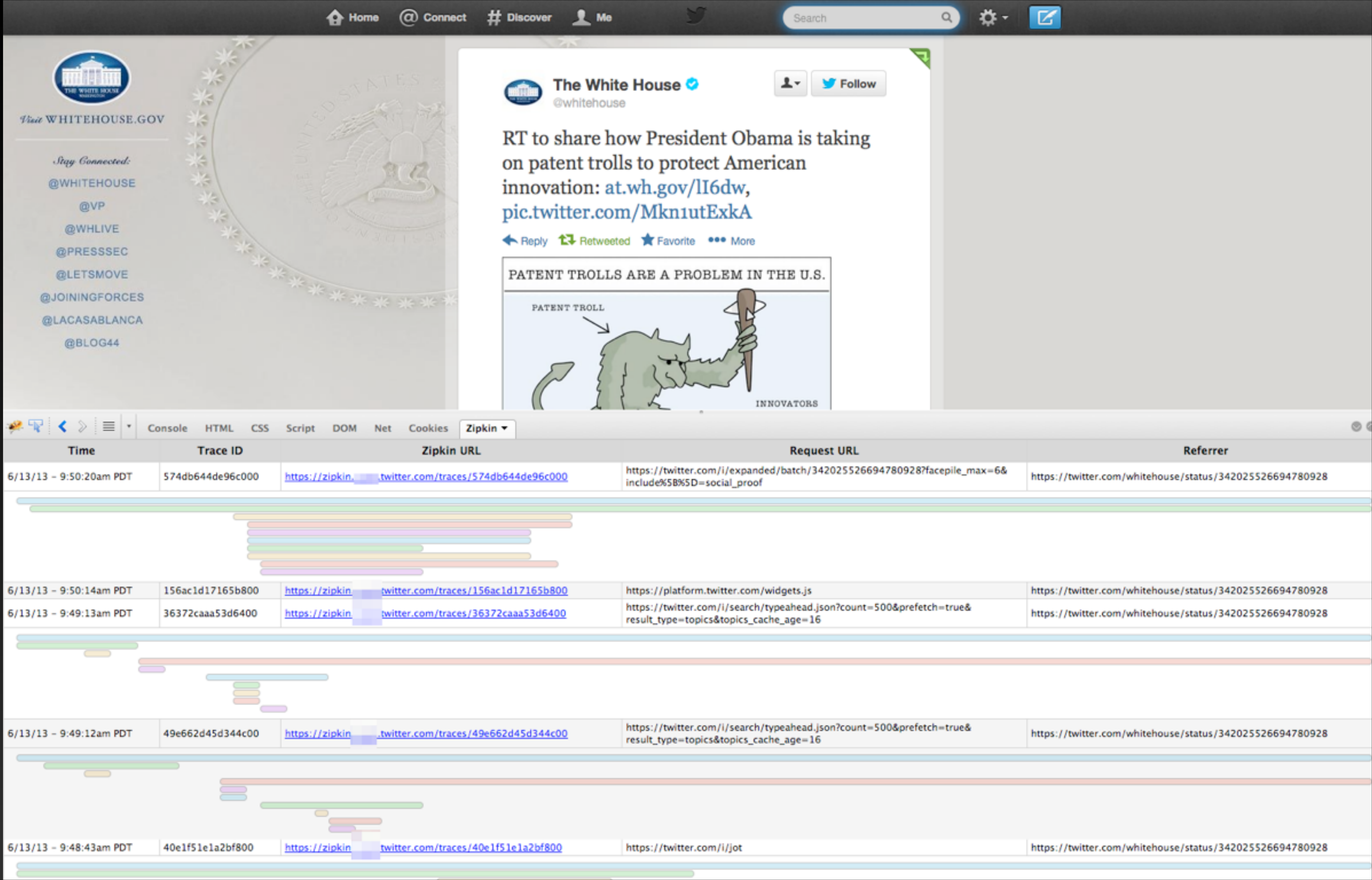
Calls test method, if present adds random *trace ID* and *span ID* to future messages on the connection

Also for HTTP

Force debug capability

Now with Firefox plugin!

https://blog.twitter.com/2013/zippy-traces-zipkin-your-browser



Home
Connect
Discover
Me
Search
Visit WHITEHOUSE.GOV
Stay Connected:
@WHITEHOUSE
@VP
@WHLIVE
@PRESSSEC
@LETSMOVE
@JOININGFORCES
@LACASABLANCA
@BLOG44
The White House
@whitehouse
Follow
RT to share how President Obama is taking on patent trolls to protect American innovation: at.wh.gov/lI6dw, pic.twitter.com/Mkn1utExkA
Reply
Retweeted
Favorite
More
PATENT TROLLS ARE A PROBLEM IN THE U.S.
PATENT TROLL
INNOVATORS
Console
HTML
CSS
Script
DOM
Net
Cookies
Zipkin
Time | Trace ID | Zipkin URL | Request URL | Referrer
6/13/13 - 9:50:20am PDT | 574db644de96c000 | https://zipkin. twitter.com/traces/574db644de96c000 | https://twitter.com/i/expanded/batch/342025526694780928?facepile_max=6&include%5B%5D=social_proof | https://twitter.com/whitehouse/status/342025526694780928
6/13/13 - 9:50:14am PDT | 156ac1d17165b800 | https://zipkin twitter.com/traces/156ac1d17165b800 | https://platform.twitter.com/widgets.js | https://twitter.com/whitehouse/status/342025526694780928
6/13/13 - 9:49:13am PDT | 36372caaa53d6400 | https://zipkin twitter.com/traces/36372caaa53d6400 | https://twitter.com/i/search/typeahead.json?count=500&prefetch=true&result_type=topics&topics_cache_age=16 | https://twitter.com/whitehouse/status/342025526694780928
6/13/13 - 9:49:12am PDT | 49e662d45d344c00 | https://zipkin twitter.com/traces/49e662d45d344c00 | https://twitter.com/i/search/typeahead.json?count=500&prefetch=true&result_type=topics&topics_cache_age=16 | https://twitter.com/whitehouse/status/342025526694780928
6/13/13 - 9:48:43am PDT | 40e1f51e1a2bf800 | https://zipkin twitter.com/traces/40e1f51e1a2bf800 | https://twitter.com/i/jot | https://twitter.com/whitehouse/status/342025526694780928

Requires services to support tracing

Limited support outside Finagle
Contributions welcome!

Thanks!

Yann Ramin
Observability @ Twitter

@theatrus
yann@twitter.com