Engineering for Visibility with Open Source Tools

Roanoke Code Camp, May 17, 2014

About Me

http://www.tildedave.com/

In this Talk

What is Visibility?

What Open Source Tools Can You Use For It?

Demo Time With Docker

Also In this Talk

What is Visibility?

Dev vs Ops

“The application does X”

“Dev”

“The application <u>really</u> does X”

“Ops”

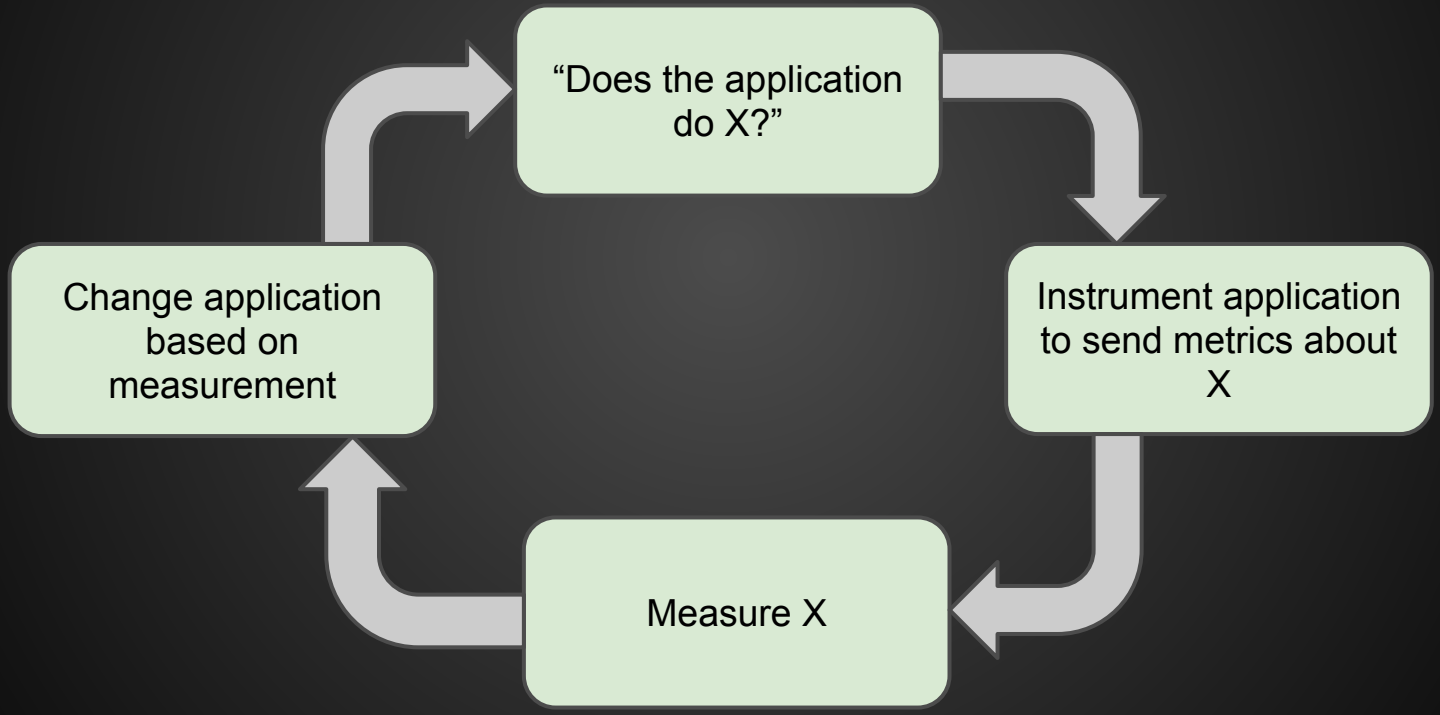
Problems With This Approach

Broken feedback loop for improvement of product

Developers and Operations have different priorities

Low-quality discussions

Continuous Improvement Cycle



Visibility Answers the Question

What is your application doing right now?

Logging is Not Enough

Logging helps <u>but</u> only shows individual actions

Custom A did action Y at time t0
Custom B did action Z at time t1
Custom C did action X at time t2

Logging is Not Enough

How many X’s in the last 5 minutes?

grep -c “X” /var/log/app.log

map reduce >:(

Logging is Not Enough

encourages 1-off investigation which does not scale

encourages people being “grep wizards”

you want to look at *aggregate data*

Alerting Is Not Enough

Server alerting is only a black box approach

Database Server at load 6 -- <u>but why?</u>

App Server at CPU 90% -- <u>does this matter?</u>

Alerting Is Not Enough

Ideally you know trends in order to plan capacity

“e.g. database load steadily climbing over the last 6 months. what can we do about this?”

Timeseries Data

Metric name, value, and a time

```
logins_per_5_minutes 120 1400165858
500_responses_served 1   1399842219
```

Timeseries Data

Can be aggregated

Can be stored for historical trends

One metric can be plotted against another

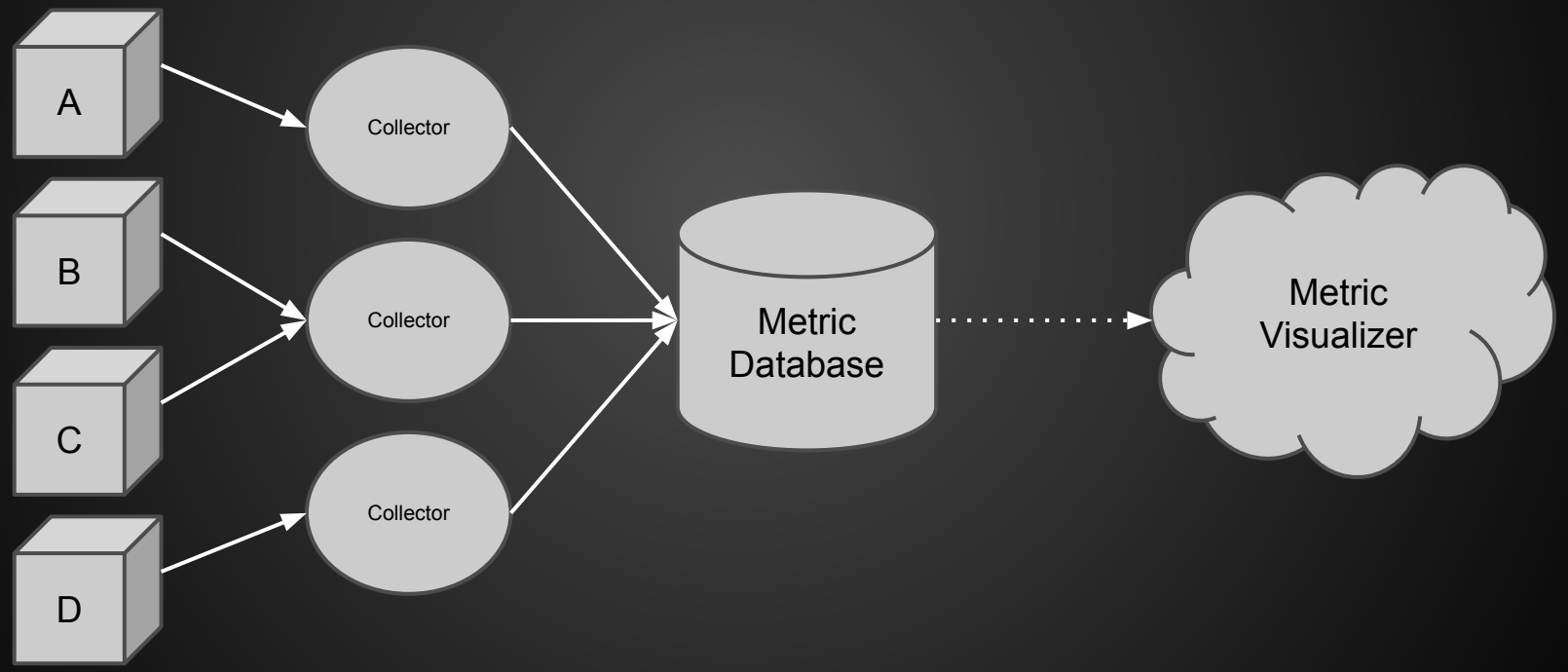
Challenges with Timeseries Data

You may want to store a lot of data at once

You may want to intake a lot of data at once

How can you use timeseries data to enable improvement?

Using Timeseries Data



The Rest of This Talk

Building Your Own Timeseries Data Collector using Open Source Software

Infrastructural Visibility

Details about technologies used for deployment

- CPU
- Memory
- Load
- # of threads connected to database
- # of slow queries
- amount of time per slow query

Infrastructural Visibility

Main strategy: install open source tools that hook in to your infrastructure in one way or another

Logster, Logstash - parse application logs

Collectd - applications, infrastructure stats

Application Visibility

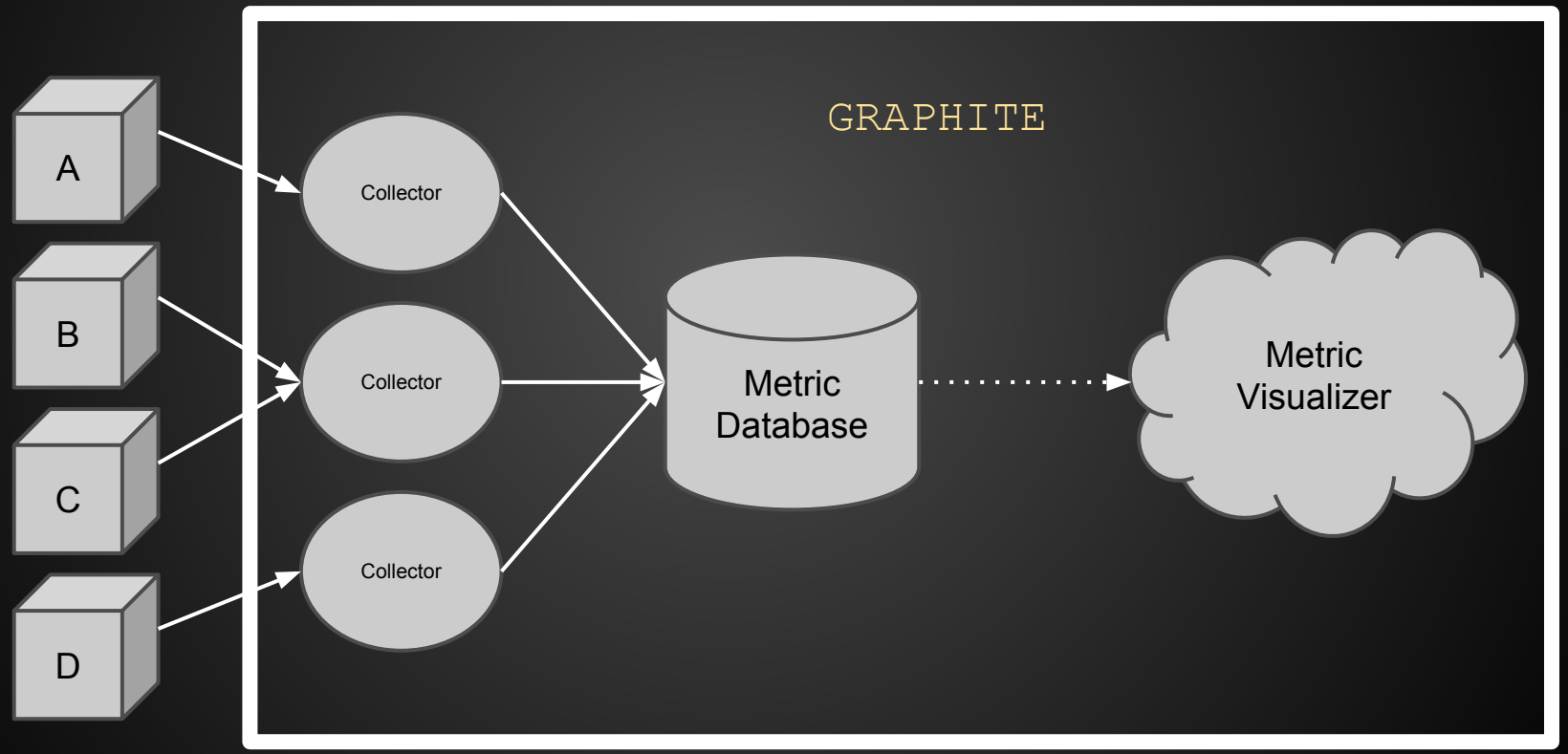
Details about custom code you've written

- time per web service call
- # of queries per a specific app server thread

Application Visibility

Main strategy: custom application code that sends metrics

Graphite: Scalable Realtime Graphing



Graphite: Scalable Realtime Graphing

Graphite Claims from their Website

“a bit of a niche application”

http://graphite.readthedocs.org/en/latest/

Orbitz: 160k different metrics/minute

Real-time graphing, even under heavy load

Graphite Claims from Me

The first tool I would install on joining a new team

Main enabler of “DevOps” continuous improvement

(Kind of a PITA to set up)

More About Graphite

Three parts:

- Carbon: Metric intake
- Whisper: Metric storage
- Web: Metric visibility

Each has its own set of config files, etc

Getting Data Into Graphite

Graphite listens on TCP port 2003

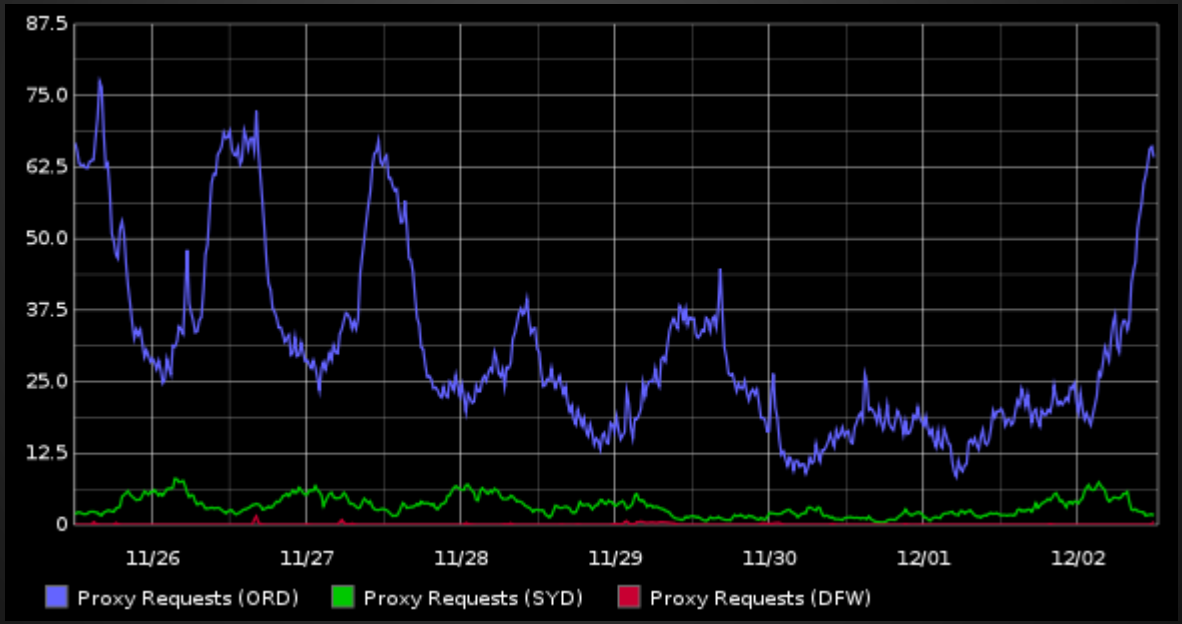
```
metric_path value timestamp
```

Other tools make this easier

- (see the rest of this talk)

Visualizing Data in Graphite

Graphs have functions on metrics



Graphite Functions

```
alias(sum(prxy-n0*.production.ord.reach.proxy_requests_total_15m_rate),
"Proxy Requests (ORD)")
alias(sum(prxy-n0*.production.syd.reach.proxy_requests_total_15m_rate),
"Proxy Requests (SYD)")
alias(sum(prxy-n0*.production.dfw.reach.proxy_requests_total_15m_rate),
"Proxy Requests (DFW)")
```

Graphite Capabilities

Graphite comes with a lot of functions for transforming/displaying data: http://graphite.readthedocs.org/en/latest/functions.html

The ones I end up using the most:

`sum, sumSeries, avgSeries, scale, highestMax, summarize`

You end up having to build a lot of knowledge about Graphite to really use it effectively.

Statsd

Stats daemon from Etsy

They wrote a blog post: http://codeascraft.com/2011/02/15/measure-anything-measure-everything/

Written in Node.js and receives stats over UDP

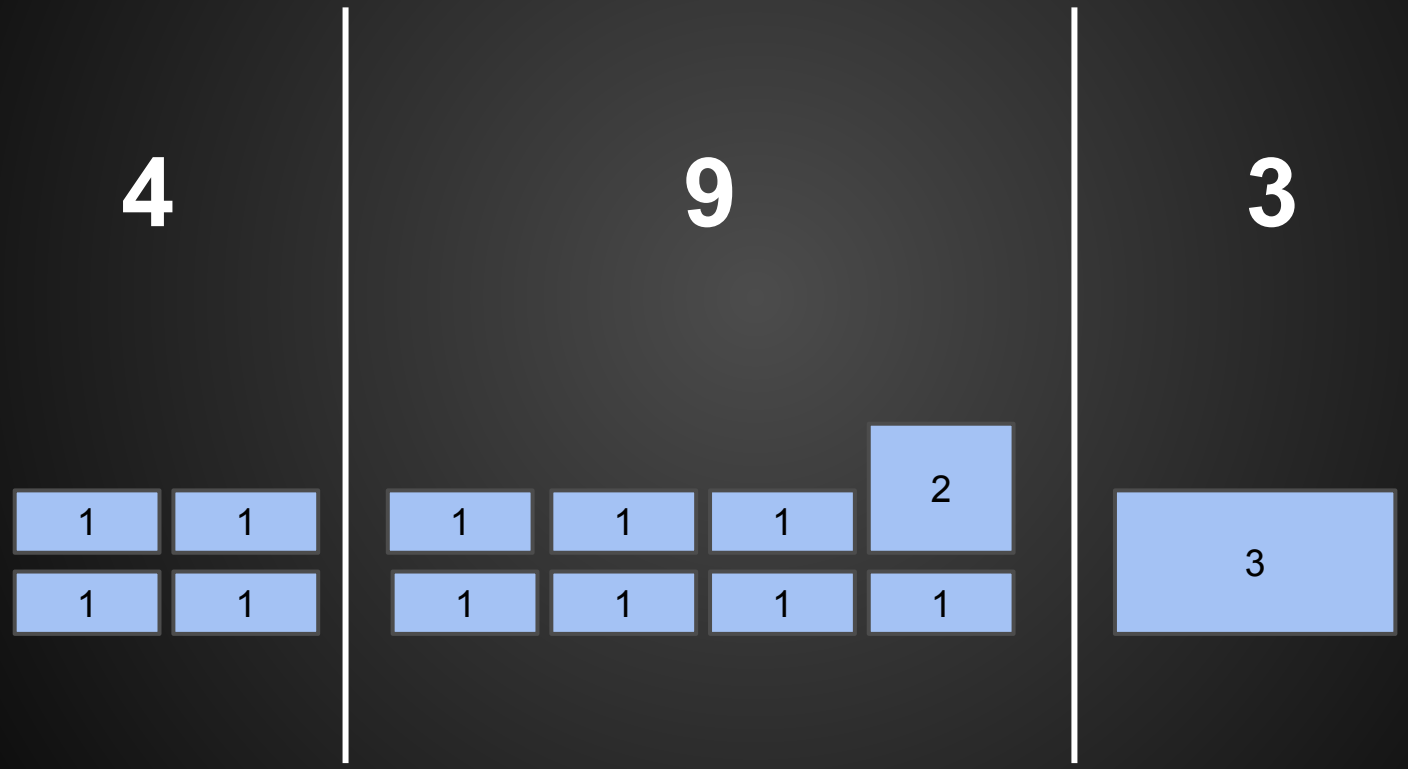
Statsd concepts

Clients send data to statsd

Every flush interval, send data to Graphite

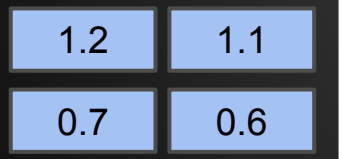
Counters, timers, and gauges have different behaviors in what data gets sent every flush interval

Statsd: Counters



Statsd: Timers

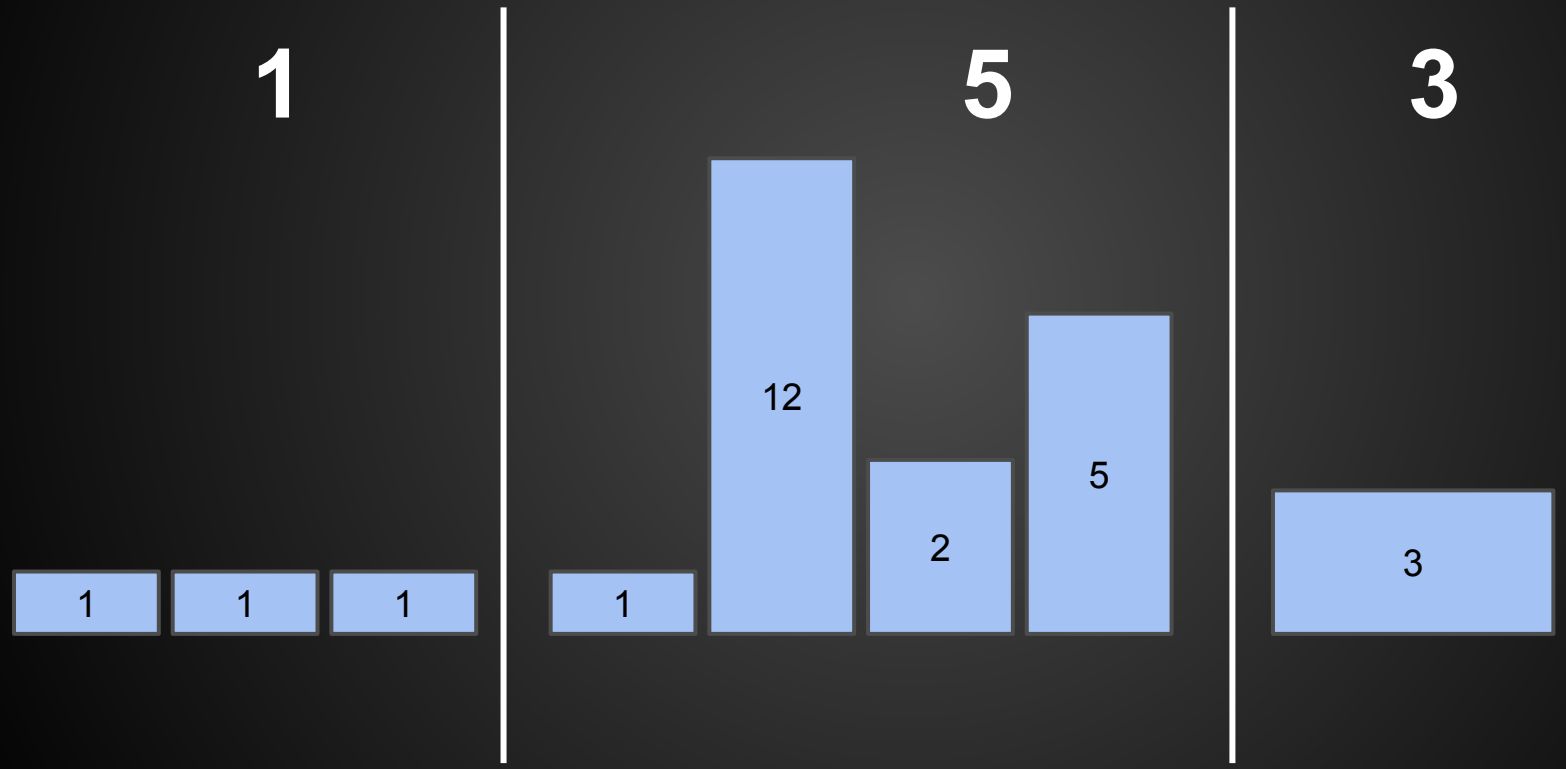
mean=0.9
min=0.6
max=1.2



mean=2.7
min=1
max=5

mean=3
min=3
max=3

Statsd: Gauges



Example Statsd Debug Output

```
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes:41.0|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:28111.0|ms
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes:1069.0|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:222.0|ms
```

The Metric Name

```
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes.41.0|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:28111.0|ms
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes:1069.0|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:222.0|ms
```

lb-n01_staging_dfw_reach_rackspace_net.apache.response.200

The Metric Type

```
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes:41.0|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:28111.0|ms
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes:1069.0|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:222.0|ms
```

Increment the counter

`lb-n01_staging_dfw_reach_rackspace_net.apache.response.200`

By the value

`1`

Another counter

```
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes:41.0|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:28111.0|ms
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes:1069.0|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:222.0|ms
```

Increment the counter

`lb-n01_staging_dfw_reach_rackspace_net.apache.bytes`

By the value

`41`

A Timer

```
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes.41.0|c
2 Dec 17:03:26 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:28111.0|ms
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.200:1|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.bytes:1069.0|c
2 Dec 17:03:28 - DEBUG: lb-n01_staging_dfw_reach_rackspace_net.apache.response.time:222.0|ms
```

The timer

`lb-n01_staging_dfw_reach_rackspace_net.apache.response`

Had the value

`28111 ms`

Disclaimer: (this is actually in microseconds from the Apache logs. statsd does not care about the unit of measure)

Sending Data to Statsd

Many client libraries for your favorite language

Node: https://github.com/msiebuhr/node-statsd-client

Java: https://github.com/youdevise/java-statsd-client

.NET: https://github.com/robbihun/NStatsD.Client

Ruby: https://github.com/reinh/statsd/

Python: https://github.com/WoLpH/python-statsd

Sending Data to Statsd

```
with statsd.timer('cassandra.query'):
    if not connection.session:
        statsd.incr('cassandra.query.session_create')
        connection.connect()

    query_statement = StatementQuery(query_string)
    with statsd.timer('cassandra.query.session_execute'):
        result = connection.session.execute(
            query_statement,
            parameters=parameters or {}
        )

    statsd.incr('cassandra.query')

    return result
```

Getting Data To Statsd

Statsd clients are great if you control the code

What if you don’t?

You are not going to patch Apache/IIS/etc to make statsd client calls

Logster

https://github.com/etsy/logster

Parse log lines and output to Graphite/Statsd

Essential complexity: watching your log files for changes

Logster

```
sudo /usr/bin/logster \
    --output=graphite \
    --graphite-host=graphite.example.com:2003 \
    SampleLogster \
    /var/log/httpd/access_log
```

Logster

```
sudo /usr/bin/logster \
   --output=graphite \
    --graphite-host=graphite.example.com:2003 \
    SampleLogster \
    /var/log/httpd/access_log
```

Type of Output
(graphite, statsd, etc)

Logster

Destination

```
sudo /usr/bin/logster \
    --output=graphite \
    --graphite-host=graphite.example.com:2003 \
    SampleLogster \
    /var/log/httpd/access_log
```

Logster

```
sudo /usr/bin/logster \
    --output=graphite \
    --graphite-host=graphite.example.com:2003 \
    SampleLogster \
    /var/log/httpd/access_log
```

Parser Class
(written in Python)

Logster

Log File to Watch

```
sudo /usr/bin/logster \
    --output=graphite \
    --graphite-host=graphite.example.com:2003 \
    SampleLogster \
    /var/log/httpd/access_log
```

Example Logster Parser

```
class SampleLogster(LogsterParser):

    def __init__(self, option_string=None):
        '''Initialize any data structures or variables needed for keeping track
        of the tasty bits we find in the log we are parsing.'''
        self.http_1xx = 0
        self.http_2xx = 0
        self.http_3xx = 0
        self.http_4xx = 0
        self.http_5xx = 0

        # Regular expression for matching lines we are interested in, and capturing
        # fields from the line (in this case, http_status_code).
        self.reg = re.compile('.*HTTP/1.\d\" (?P<http_status_code>\d{3}) .*')
```

Example Logster Parser

```
def parse_line(self, line):
    '''This function should digest the contents of one line at a time, updating
    object's state variables. Takes a single argument, the line to be parsed.'''

    try:
        # Apply regular expression to each line and extract interesting bits.
        regMatch = self.reg.match(line)

        if regMatch:
            linebits = regMatch.groupdict()
            status = int(linebits['http_status_code'])

            if (status < 200):
                self.http_1xx += 1
            elif (status < 300):
                self.http_2xx += 1
            elif (status < 400):
                self.http_3xx += 1
            elif (status < 500):
                self.http_4xx += 1
            else:
                self.http_5xx += 1
```

Example Logster Parser

```
def get_state(self, duration):
    '''Run any necessary calculations on the data collected from the logs
    and return a list of metric objects.'''
    self.duration = duration

    # Return a list of metrics objects
    return [
        MetricObject("http_1xx", (self.http_1xx / self.duration), "Responses per sec"),
        MetricObject("http_2xx", (self.http_2xx / self.duration), "Responses per sec"),
        MetricObject("http_3xx", (self.http_3xx / self.duration), "Responses per sec"),
        MetricObject("http_4xx", (self.http_4xx / self.duration), "Responses per sec"),
        MetricObject("http_5xx", (self.http_5xx / self.duration), "Responses per sec"),
    ]
```

Logstash Agent

Heavier weight than Logster

Parse events (log messages), output metrics

http://logstash.net/docs/1.4.1/outputs/statsd
http://logstash.net/docs/1.4.1/outputs/graphite

Logstash Agent

We (Rackspace) use Rackspace Cloud Monitoring

Run Cloud Monitoring Agent on all machines

- checks CPU, I/O, disk system, etc

How to get metrics out?

Logstash Agent

Fri Jan 10 11:48:29 2014 DBG: 50.57.61.12:443 (hostname=agent-endpoint-ord.monitoring.api.rackspacecloud.com connID=34) -> SENDING: (endpoint:44391) => {"target":"endpoint","source":"848592c9-0130-445a-c450-bc764e111acb","id":"44391","params":{"timestamp":1389354509170,"status":"success","state":"available","check_type":"agent.load_average","metrics":[[null,{"15m":{"t":"double","v":"0.27"},"5m":{"t":"double","v":"0.33"},"1m":{"t":"double","v":"0.36"}}]],"check_id":"ch96OT6akx"},"v":"1","method":"check_metrics.post"}

Logstash Agent

```
input {
  file {
    'path' => '/var/log/rackspace-monitoring-agent.log'
    'type' => 'rackspace-monitoring-agent'
  }
}
```

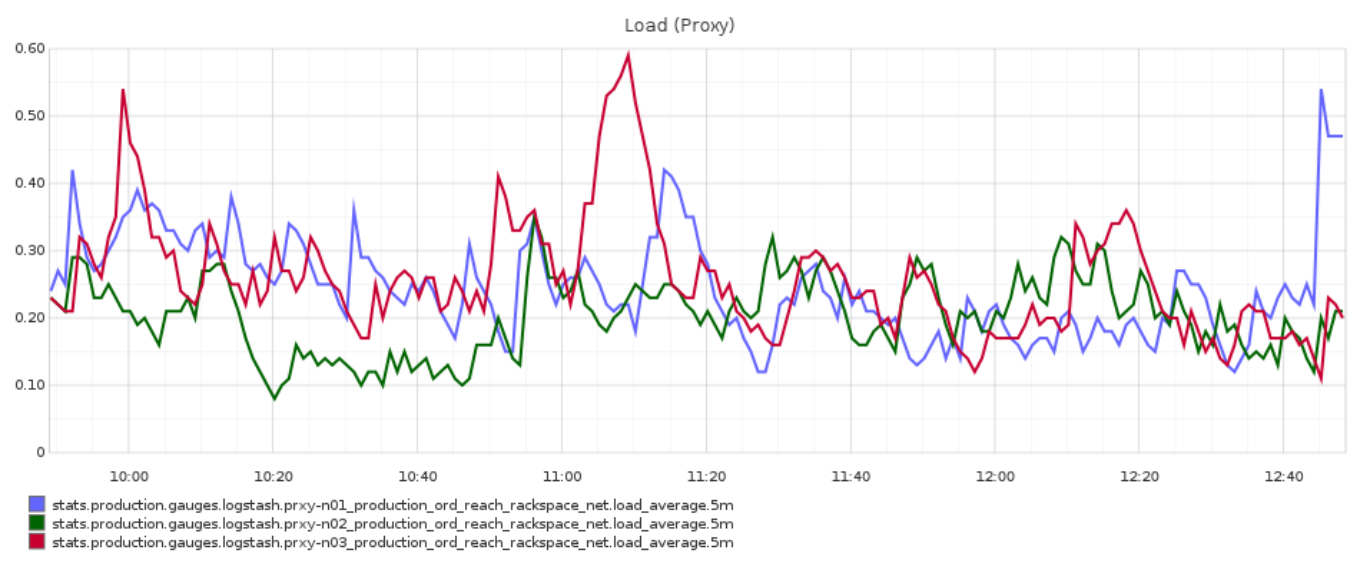
Logstash Agent

```
filter {
  grep {
    'add_tag' => ['agent.load_average']
    'drop' => false
    'match' => ['message', 'check_type":"agent.load_average"']
    'type' => 'rackspace-monitoring-agent'
  }

  grok {
    'patterns_dir' => '/opt/logstash/agent/etc/patterns'
    'pattern' => ['message', '.*SENDING.*=> %{GREEDYDATA:raw_json_data}.*']
    'type' => 'rackspace-monitoring-agent'
  }

  json {
    'source' => 'raw_json_data'
    'target' => 'sent_data'
    'type' => 'rackspace-monitoring-agent'
  }
}
```

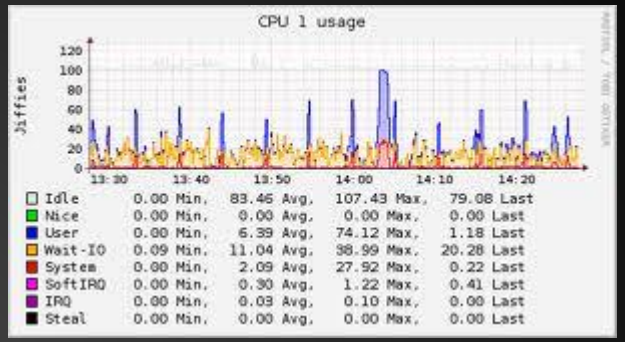
Logstash Agent

```
output {
  statsd {
    'gauge' => ['load_average.15m',
                '%{[sent_data][params][metrics][0][1][15m][v]}',
                'load_average.5m',
                '%{[sent_data][params][metrics][0][1][5m][v]}',
                'load_average.1m',
                '%{[sent_data][params][metrics][0][1][1m][v]}']
    'tags' => ['agent.load_average']
  }
}
```

Logstash Agent



Collectd



Metric collection daemon

Plugins to read data from many common apps

Plugins to output data to Statsd/Graphite/etc

Collectd

Plugins for:

- memory, cpu, load average
- nginx, java, apache
- output to graphite, statsd, etc

Collectd

Name	Type	Manpage	Available since
AMQP plugin	Read, Write	collectd.conf(5)	5.0
Apache plugin	Read	collectd.conf(5)	3.9
APC UPS plugin	Read	collectd.conf(5)	3.10
Apple Sensors plugin	Read		3.9
Aquaero plugin	Read	collectd.conf(5)	5.4
Ascent plugin	Read	collectd.conf(5)	4.4
Battery plugin	Read		3.7
BIND plugin	Read	collectd.conf(5)	4.6
Carbon plugin	Write		4.9
cgroups plugin	Read	collectd.conf(5)	5.4
ConnTrack plugin	Read		4.7
ContextSwitch plugin	Read		4.9
CPU plugin	Read		1.3
CPUFreq plugin	Read	collectd.conf(5)	3.4
CSV plugin	Write	collectd.conf(5)	4.0
cURL plugin	Read	collectd.conf(5)	4.6
cURL-JSON plugin	Read	collectd.conf(5)	4.8
cURL-XML plugin	Read	collectd.conf(5)	4.10
DBI plugin	Read	collectd.conf(5)	4.6
DF plugin	Read	collectd.conf(5)	3.6
Disk plugin	Read	collectd.conf(5)	1.5
DNS plugin	Read	collectd.conf(5)	3.11
E-Mail plugin	Read	collectd.conf(5)	3.11
Entropy plugin	Read		4.0
Ethstat plugin	Read		5.1
Exec plugin	Read	collectd-exec(5)	4.0
FileCount plugin	Read	collectd.conf(5)	4.5
FSCache plugin	Read		4.7
GenericJMX plugin	Read	collectd-java(5)	4.8
gmond plugin	Read	collectd.conf(5)	4.7
HDDTemp plugin	Read	collectd.conf(5)	3.1
Interface plugin	Read	collectd.conf(5)	1.0
IPMI plugin	Read	collectd.conf(5)	4.4
IPTables plugin	Read	collectd.conf(5)	4.0
IPVS plugin	Read		4.2
IRQ plugin	Read	collectd.conf(5)	4.0
Java plugin	Binding	collectd.conf(5), collectd-java(5)	4.7
libvirt plugin	Read	collectd.conf(5)	4.3
Load plugin	Read		1.0
LogFile plugin	Logging	collectd.conf(5)	3.9
LPAR plugin	Read	collectd.conf(5)	5.0
LVM plugin	Read		5.4
MadWifi plugin	Read	collectd.conf(5)	4.8
MBMon plugin	Read	collectd.conf(5)	3.11
MD plugin	Read	collectd.conf(5)	5.1
memcachec plugin	Read	collectd.conf(5)	4.7
memcached plugin	Read	collectd.conf(5)	4.2
Memory plugin	Read		1.0
MIC plugin	Read	collectd.conf(5)	5.4
Modbus plugin	Read	collectd.conf(5)	4.10

Some Of My Favorite Metrics From Work

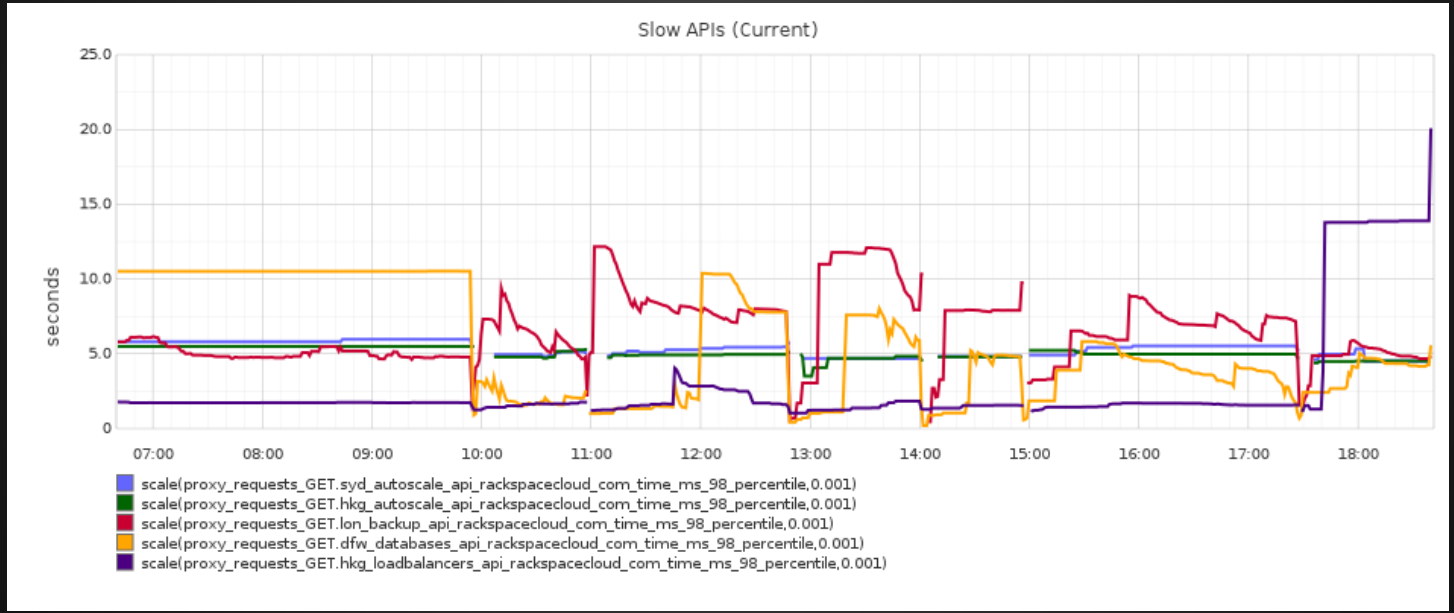
My Team

Rackspace Cloud Control Panel

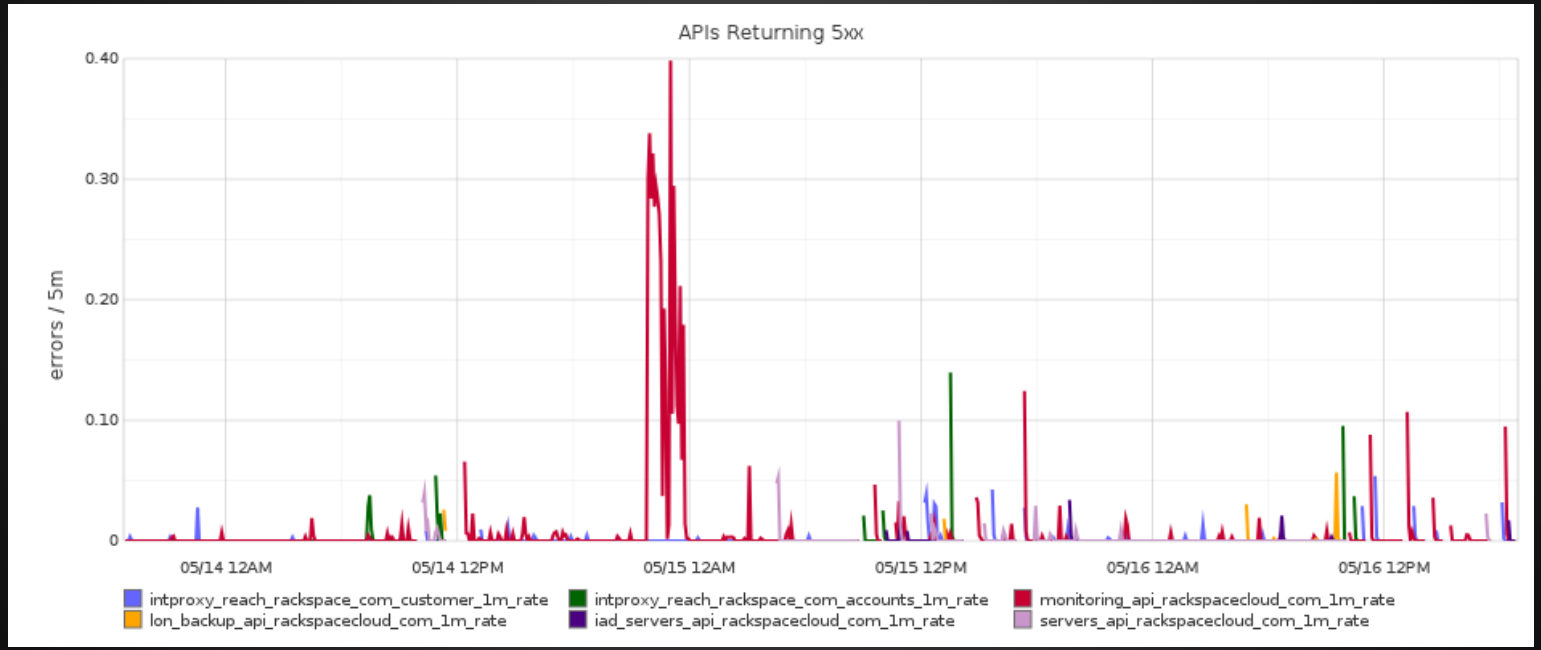
UI on top of a bunch of other services (15+)

Visibility is important (or we go insane)

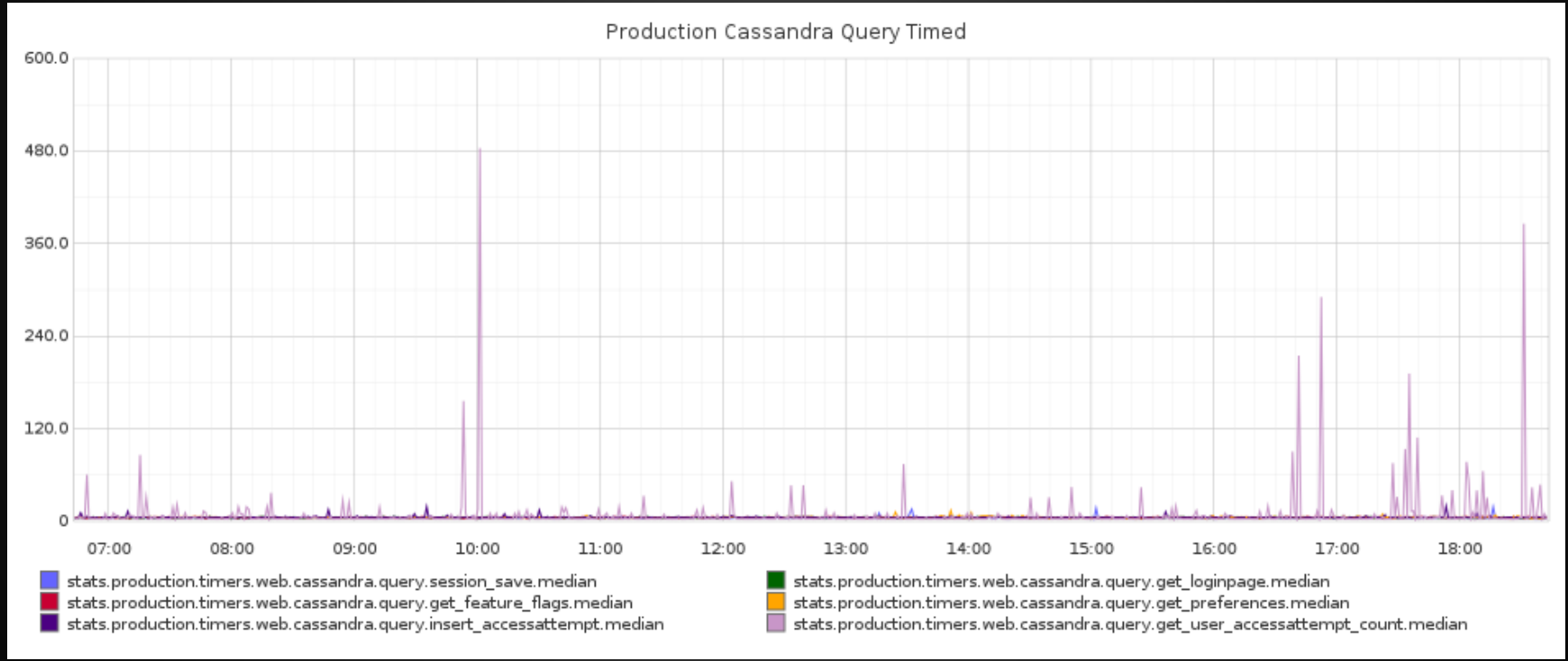
Why is the Control Panel Slow?



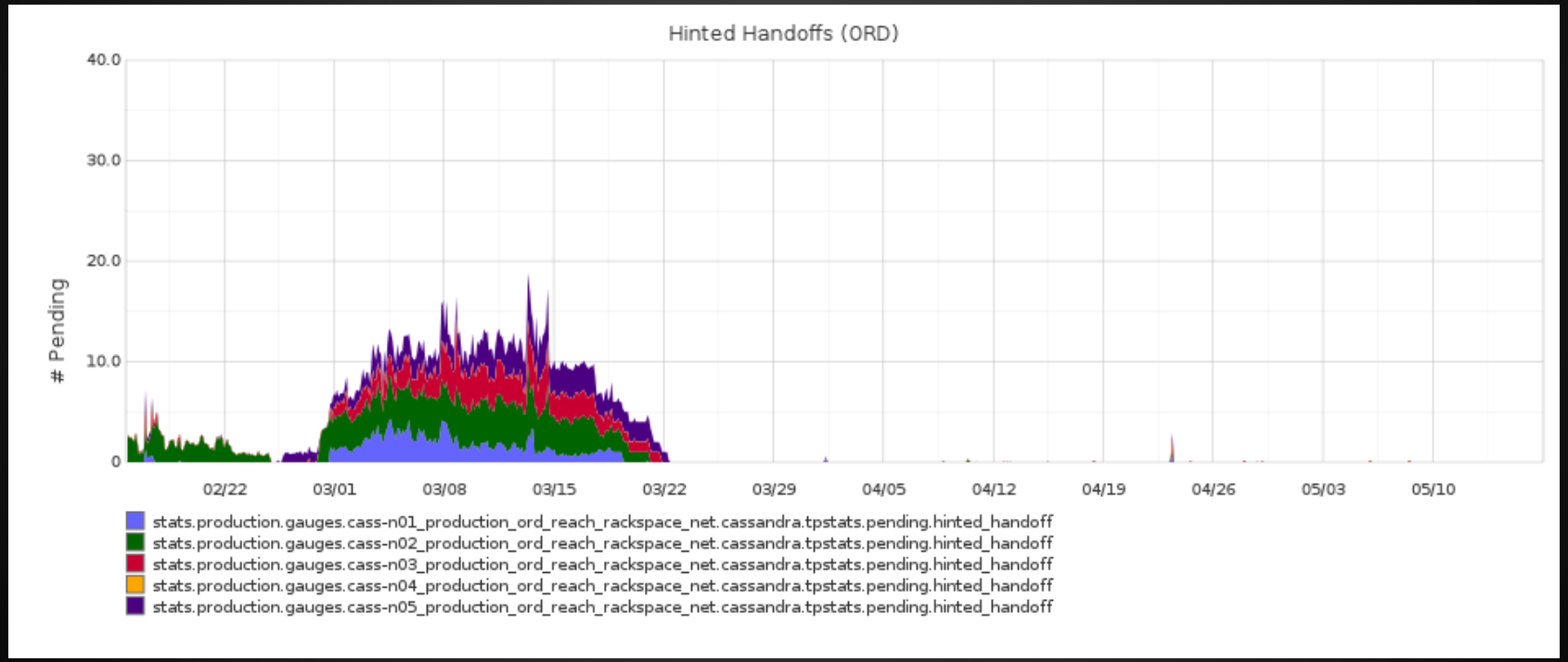
Why Is The Control Panel Erroring?



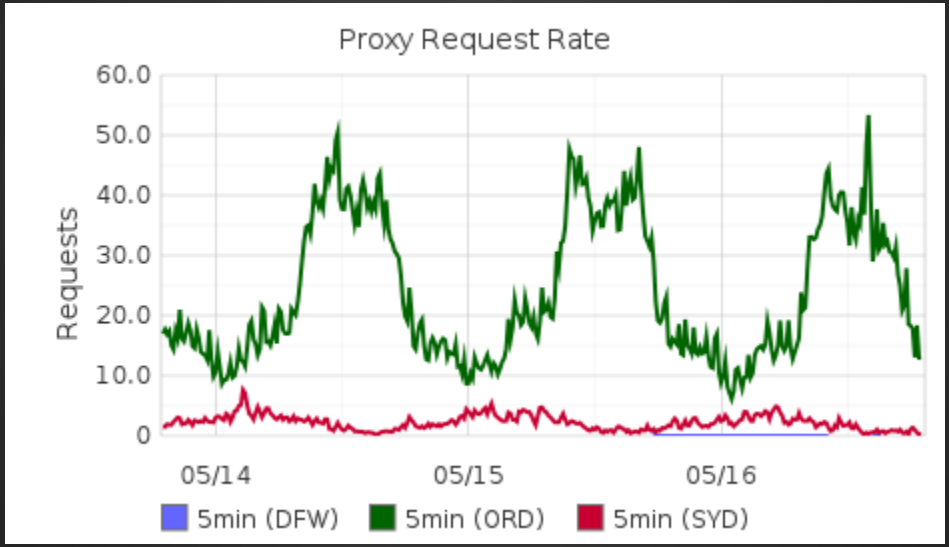
What Are Our Slowest Queries?



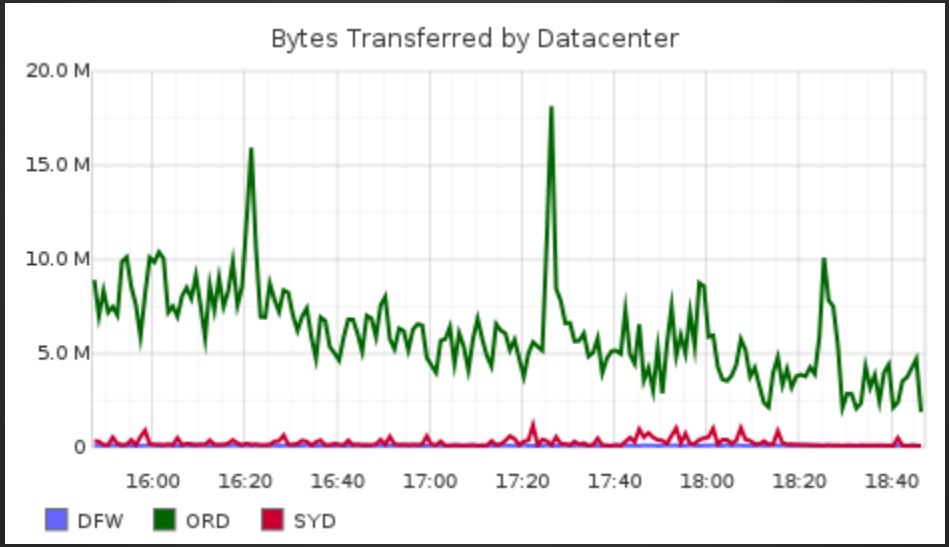
Is Cassandra Healthy?



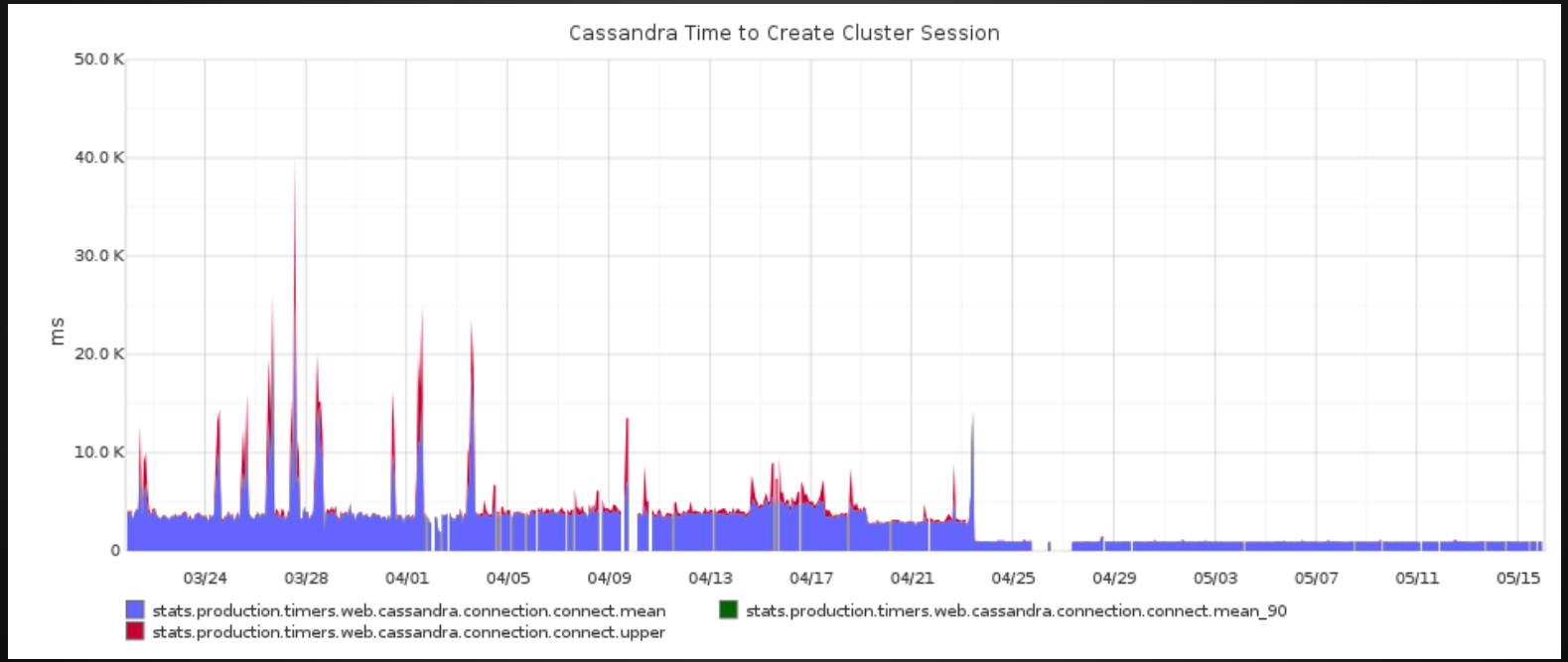
How Utilized Are Our App Serevrs?



How Much Traffic Are We Serving?



How Great Was Cassandra 2.0.7?



My Journey on Visibility

Final Thoughts

Outputting metrics in real-time enables continuous improvement

This is probably the first thing I'd set up on any future team

Final Thoughts

You can only really answer questions that you've enabled yourself to answer

Trying to reconstruct nontrivial events from logs is <u>extremely difficult</u>

Final Thoughts

Systems should be instrumented to provide information from their deployment to the product development team

This lets you have higher quality conversations and make great products

Demo Time With Docker