Application performance management with open source tools

Monica Sarbu & Tudor Golubenco
(@monicasarbu & @tudor_g)

BERLIN BUZZWORDS 2015 MAY 31—JUNE 3

Intro

- Software devs
- Worked at a startup doing a VoIP monitoring product
- Startup acquired by Acme Packet, acquired by Oracle
- Working on @packetbeat

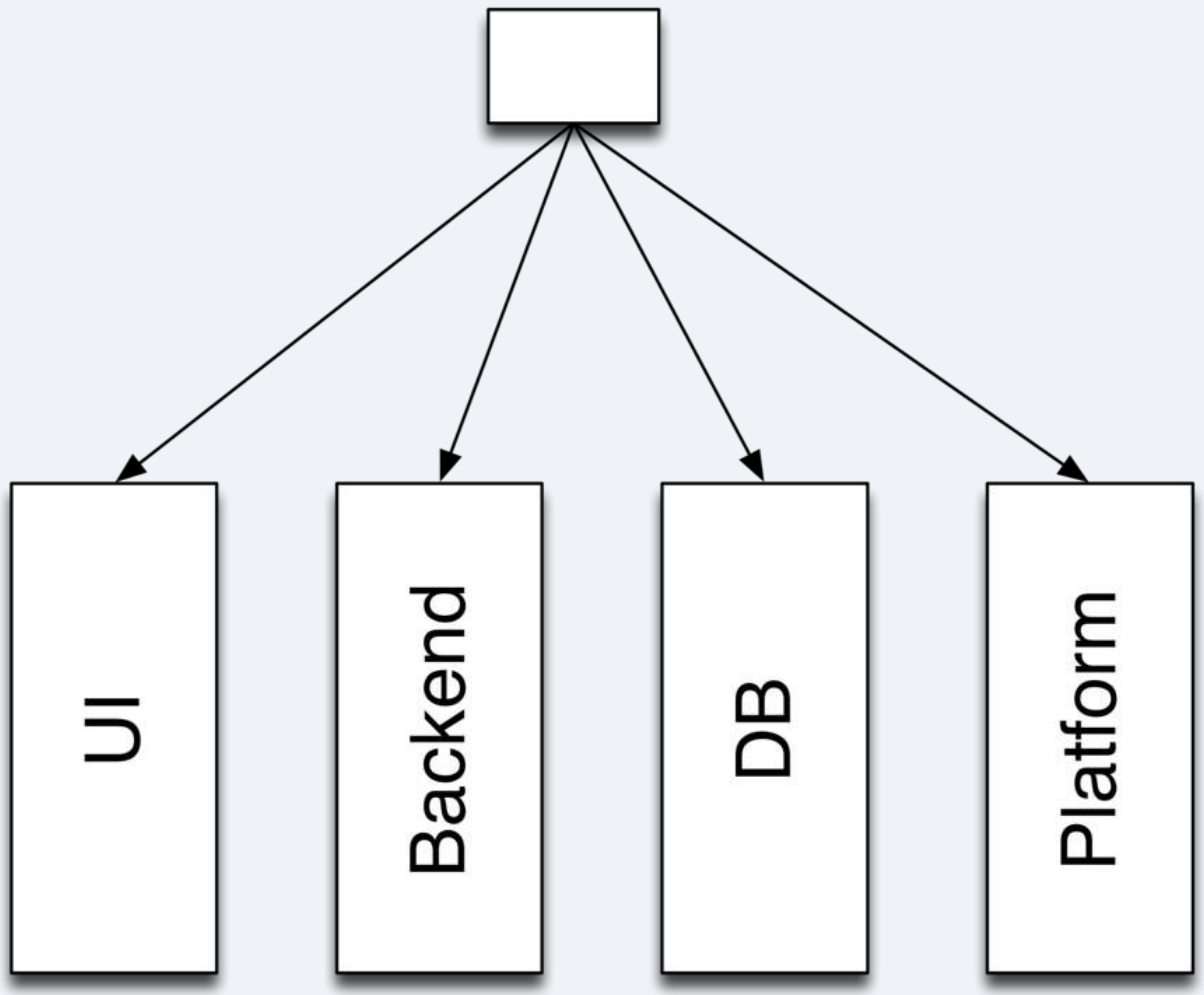
Scaling

- Infrastructure:
  - scale to 100s, 1.000s, 10.000s of servers
- Organization:
  - scale to 100s, 1.000s, 10.000s of employees

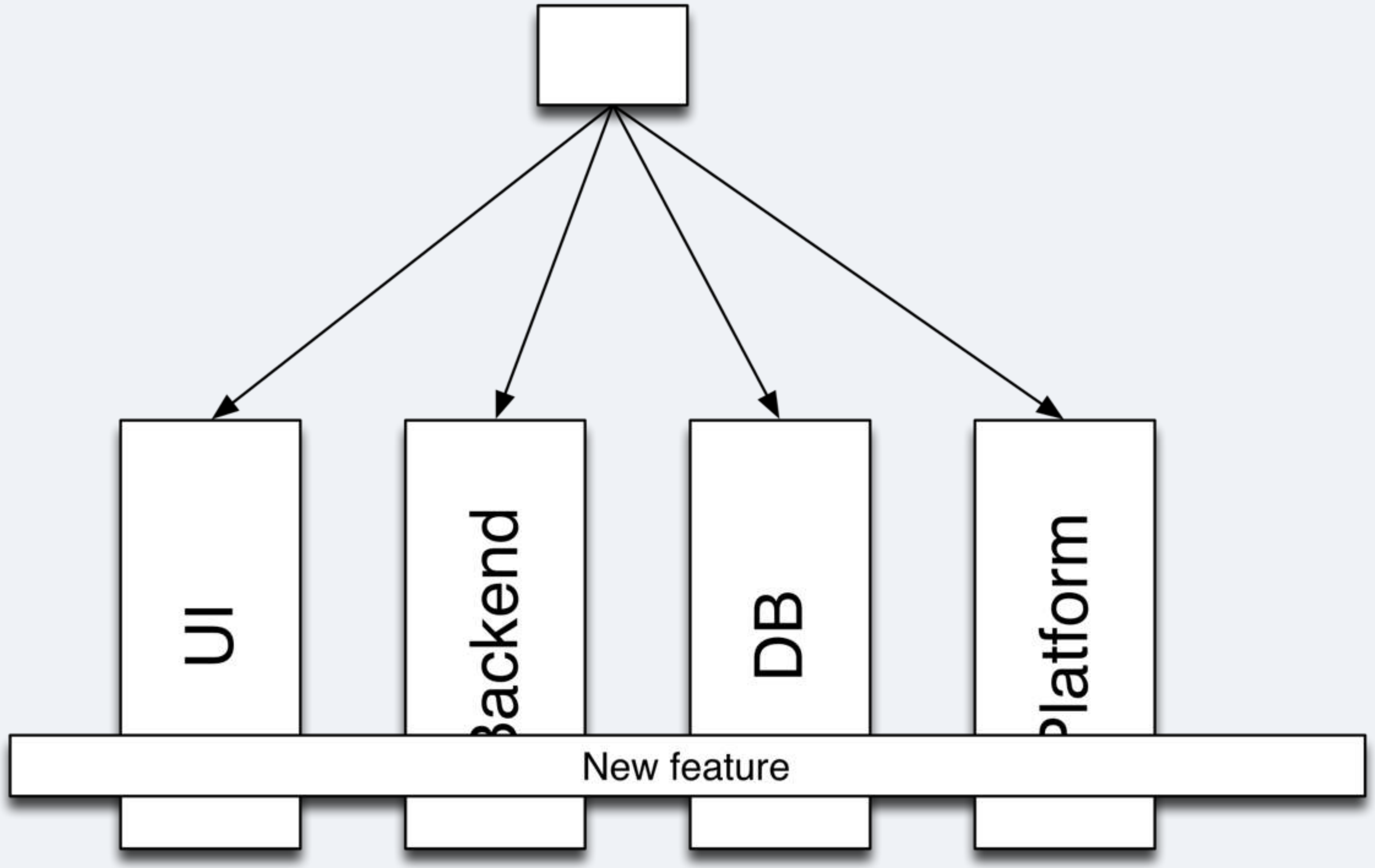
Conway's law

- *“Organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations"*

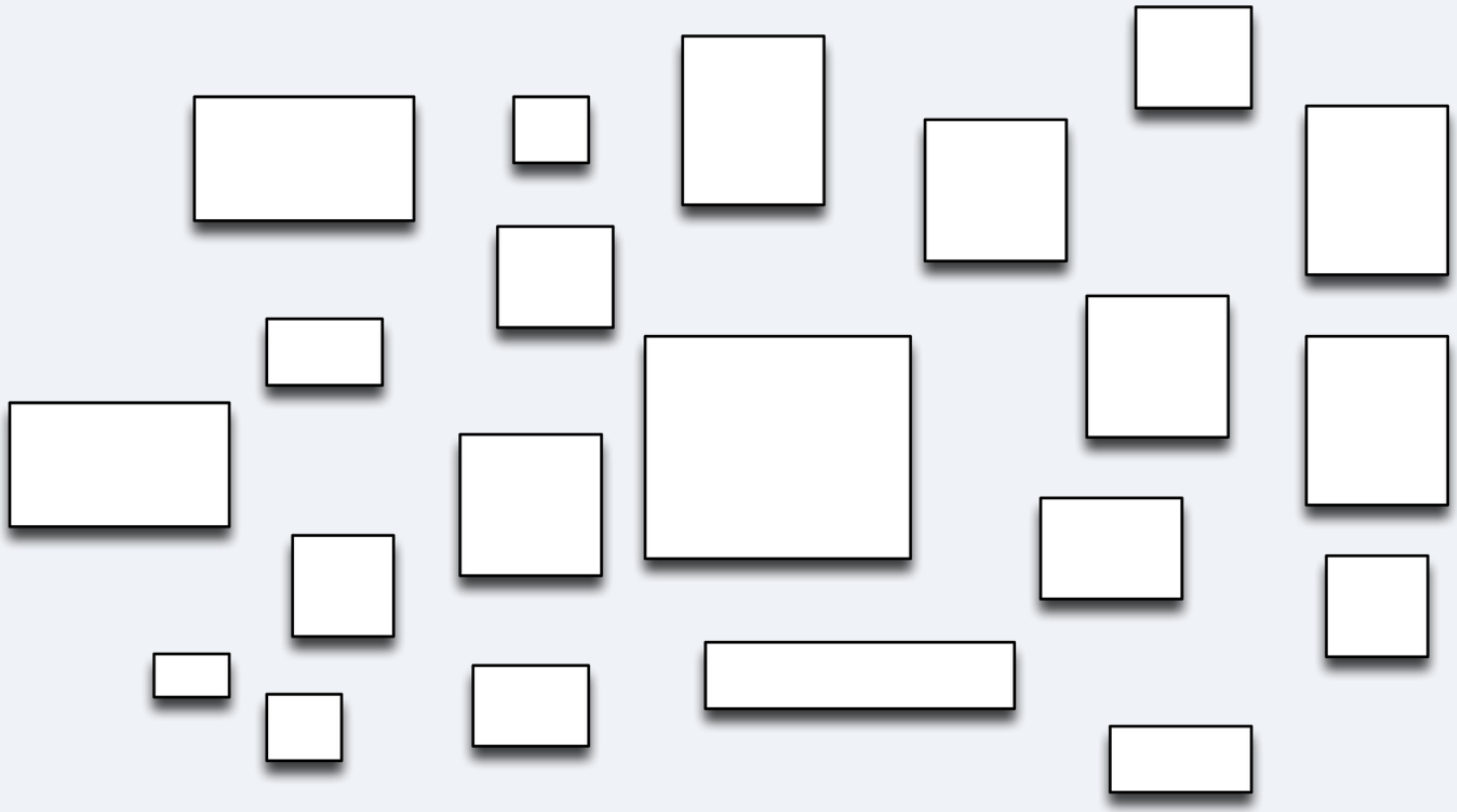
First org chart



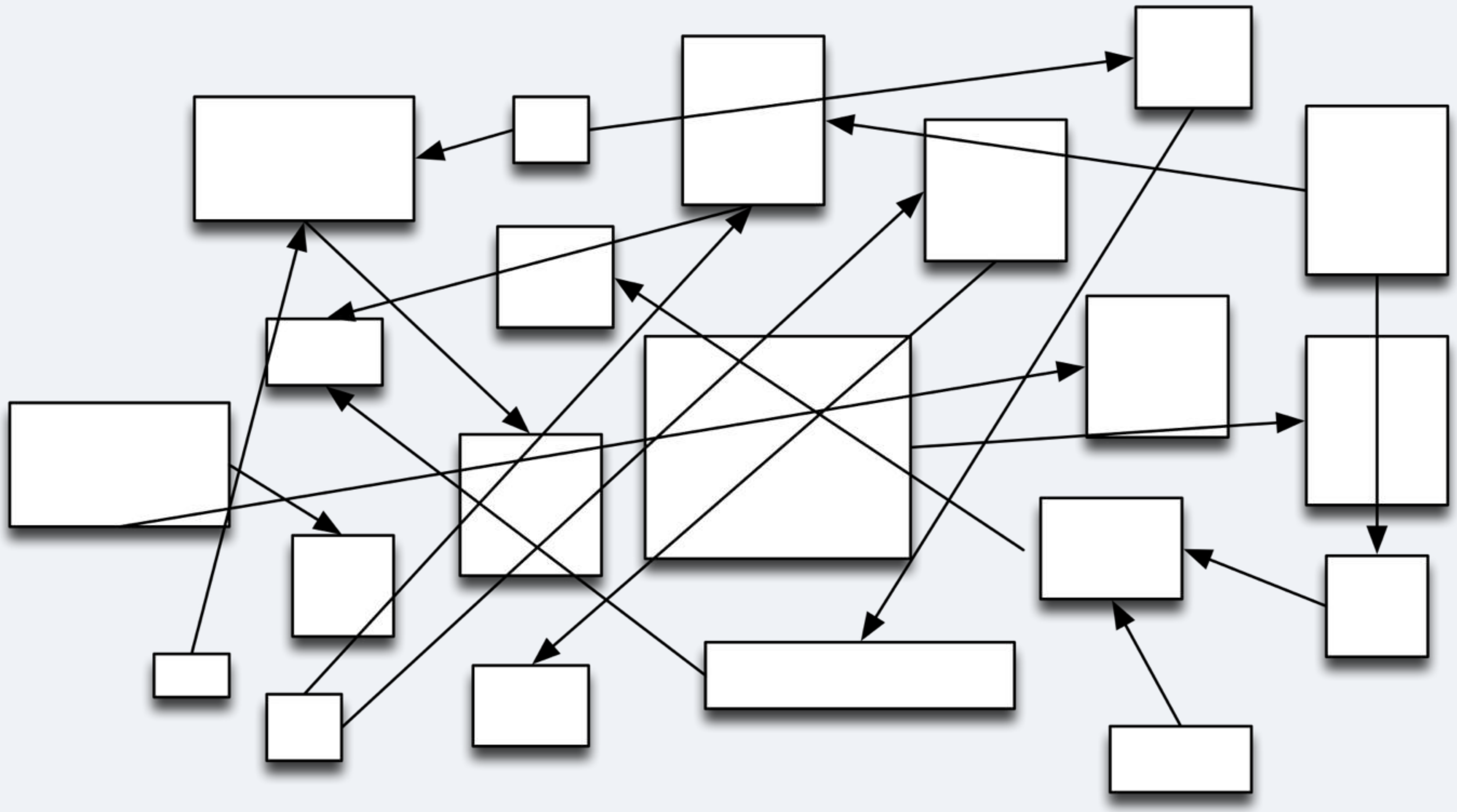
First org chart



Microservices

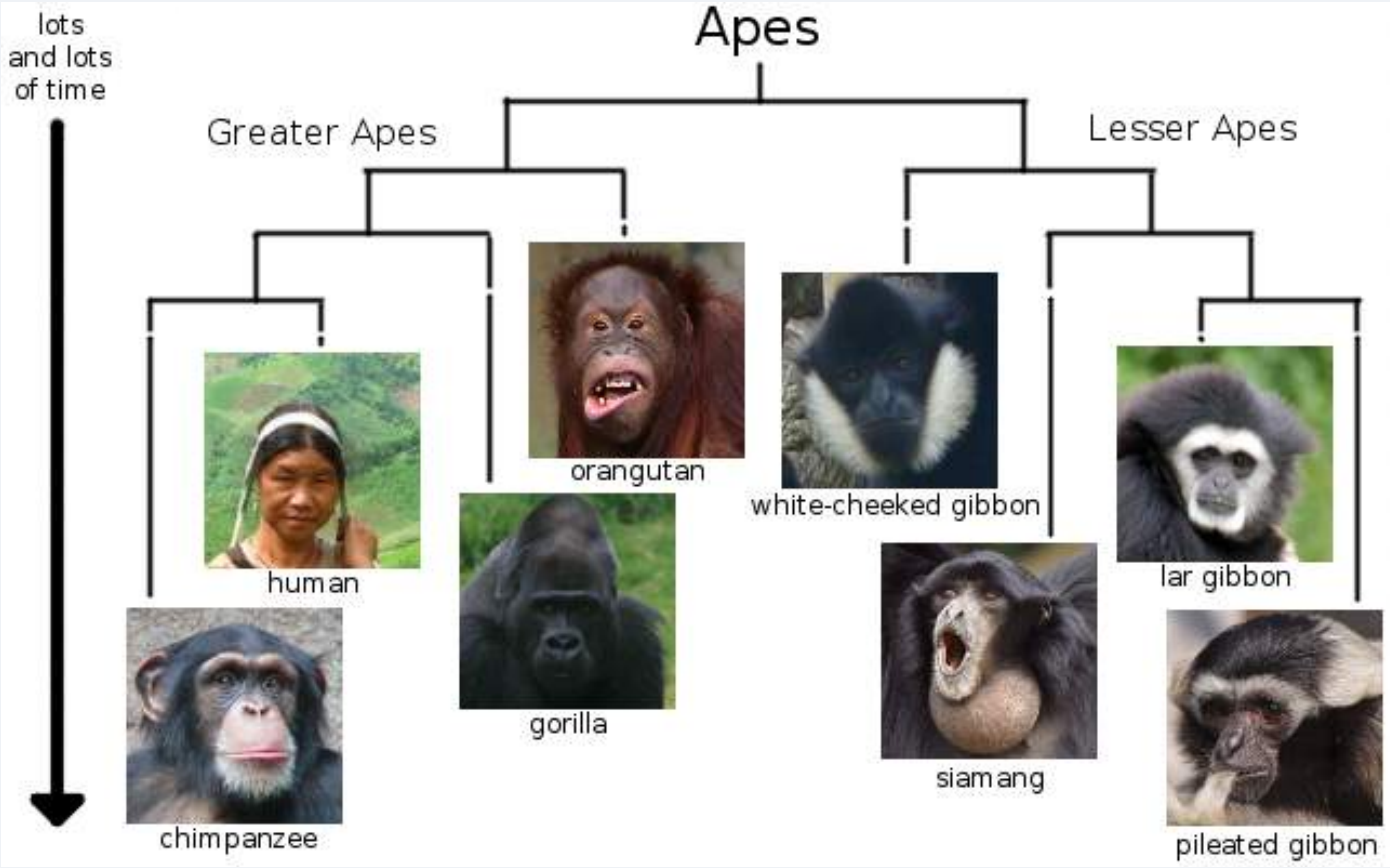


Microservices



Evolution

- Applications evolve over time
- Adapt to new requirements
- Mutations are kind of random
- You need to select the good mutations



Operational monitoring

- Critical
  - It’s how you filter out the bad mutations and keep the good ones
- Difficult
  - Highly heterogenous infrastructures
  - Show the global state of a distributed system

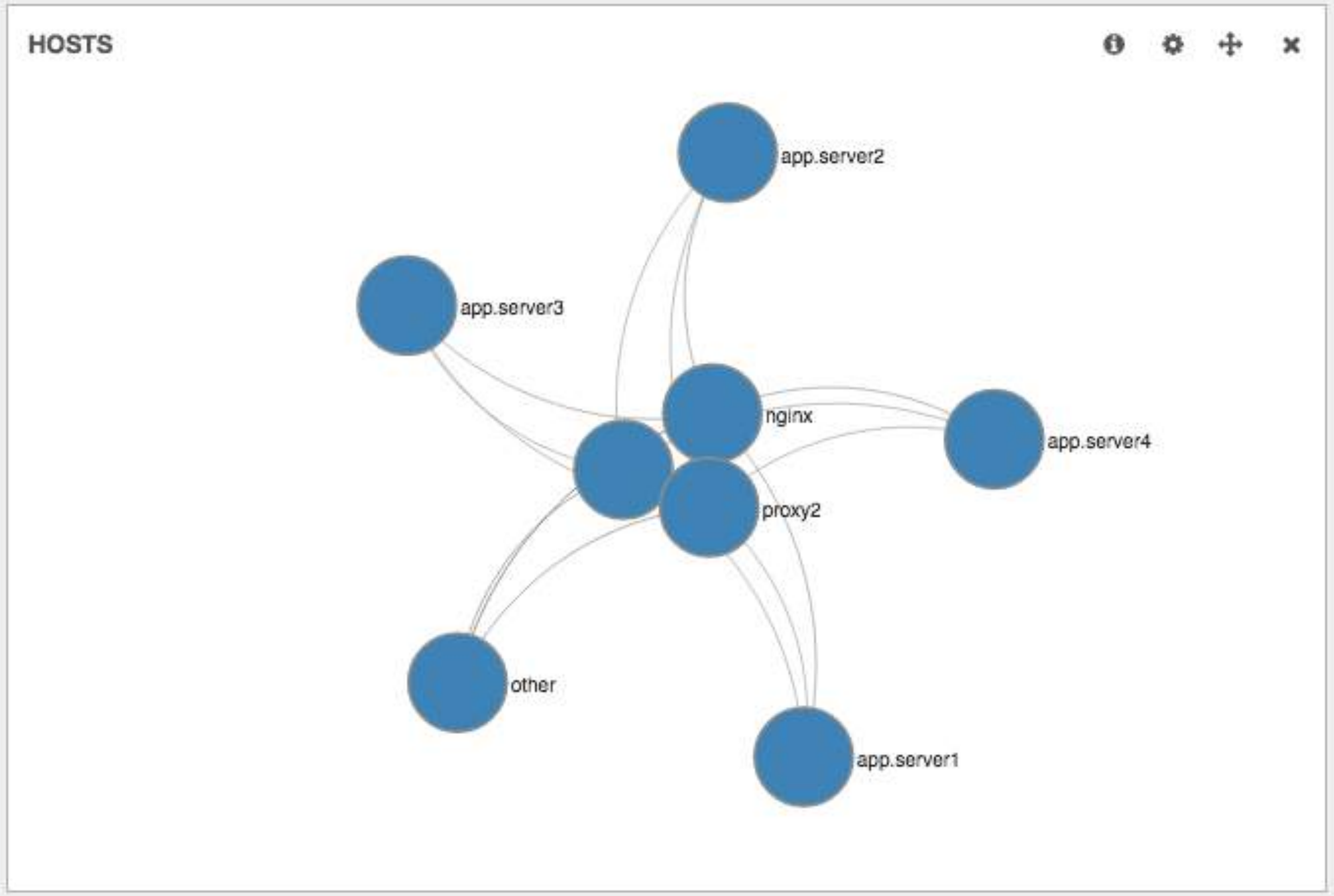
monitoring and troubleshooting distributed applications

Requirements

- Scalable and reliable
- Extract data from different sources
- Low overhead
- Low configuration
- Simple, easy to understand

Start from the communication

- The communication between components gets you the big picture
- Protocols are standard
- Packet data is objective
- No latency overhead



Packetbeat

- First public version in 05.2014
- Open Source, written in Golang

What is Packetbeat?

¯_(ツ)_/¯

Packetbeat shipper

- Running on your application servers
- Follows TCP streams, decodes upper layer protocols like HTTP, MySQL, PgSQL, Redis, Thrift-RPC, etc
- Correlates requests with responses
- Captures data and measurements from transactions and environment
- Exports data in JSON format

```
{
  "client_ip": "127.0.0.1",
  "client_port": 46981,
  "ip": “127.0.0.1",
  "query": "select * from test",
  "method": "SELECT",
  "pgsql": {
    "error_code": "",
    "error_message": "",
    "error_severity": "",
    "iserror": false,
    "num_fields": 2,
    "num_rows": 2
  },
  "port": 5432,
  "responsetime": 12,
  "bytes_out": 95,
  "status": "OK",
  "timestamp": "2015-05-27T22:27:57.409Z",
  "type": "pgsql"
}
```

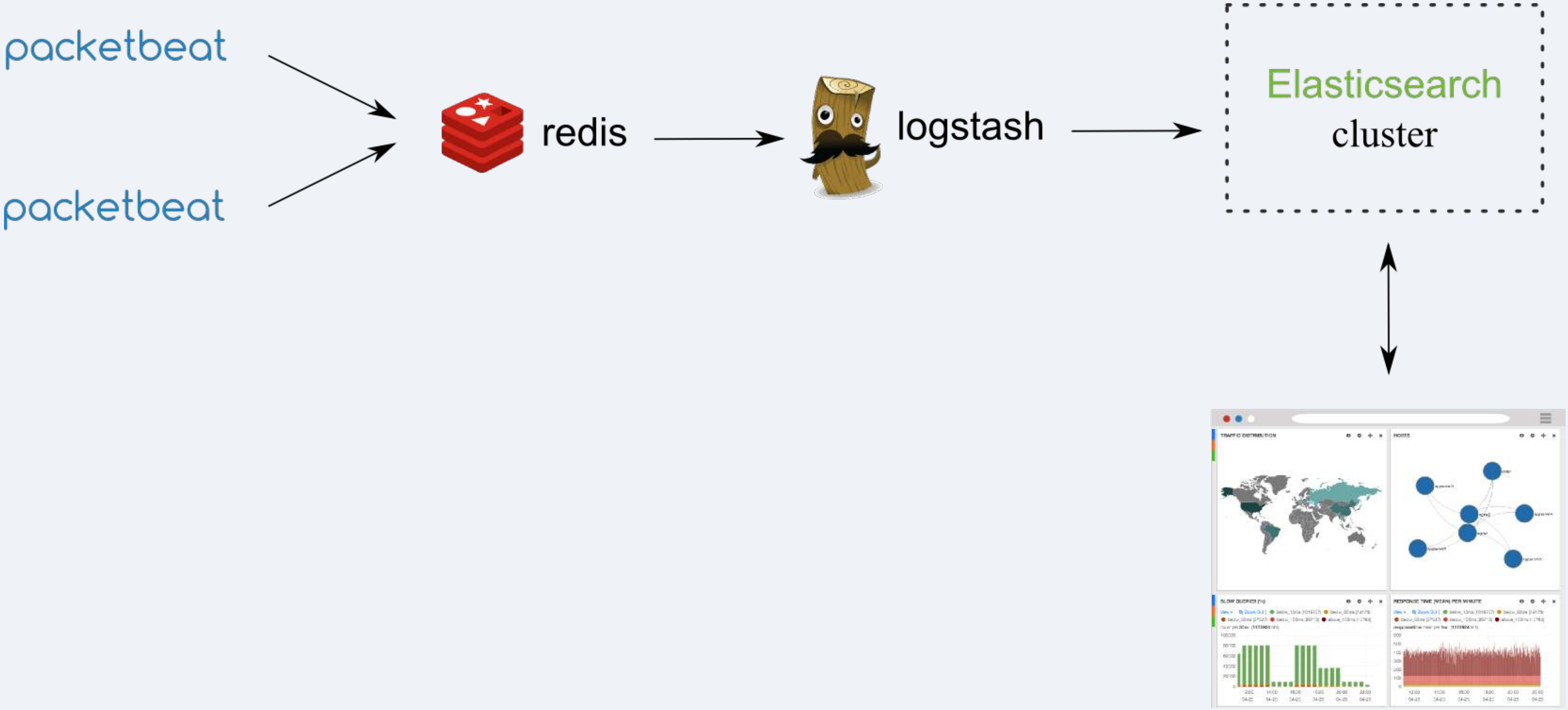
What do we do with the data?

¯\(°_o)/¯

The traditional way

- Decide what metrics you need (requests per second for each server, response time percentiles, etc.)
- Write code to extract these metrics, store them in a DB
- Store the transactions in a DB
- But:
  - Each metric adds complexity
  - Features like drilling down and top N are difficult

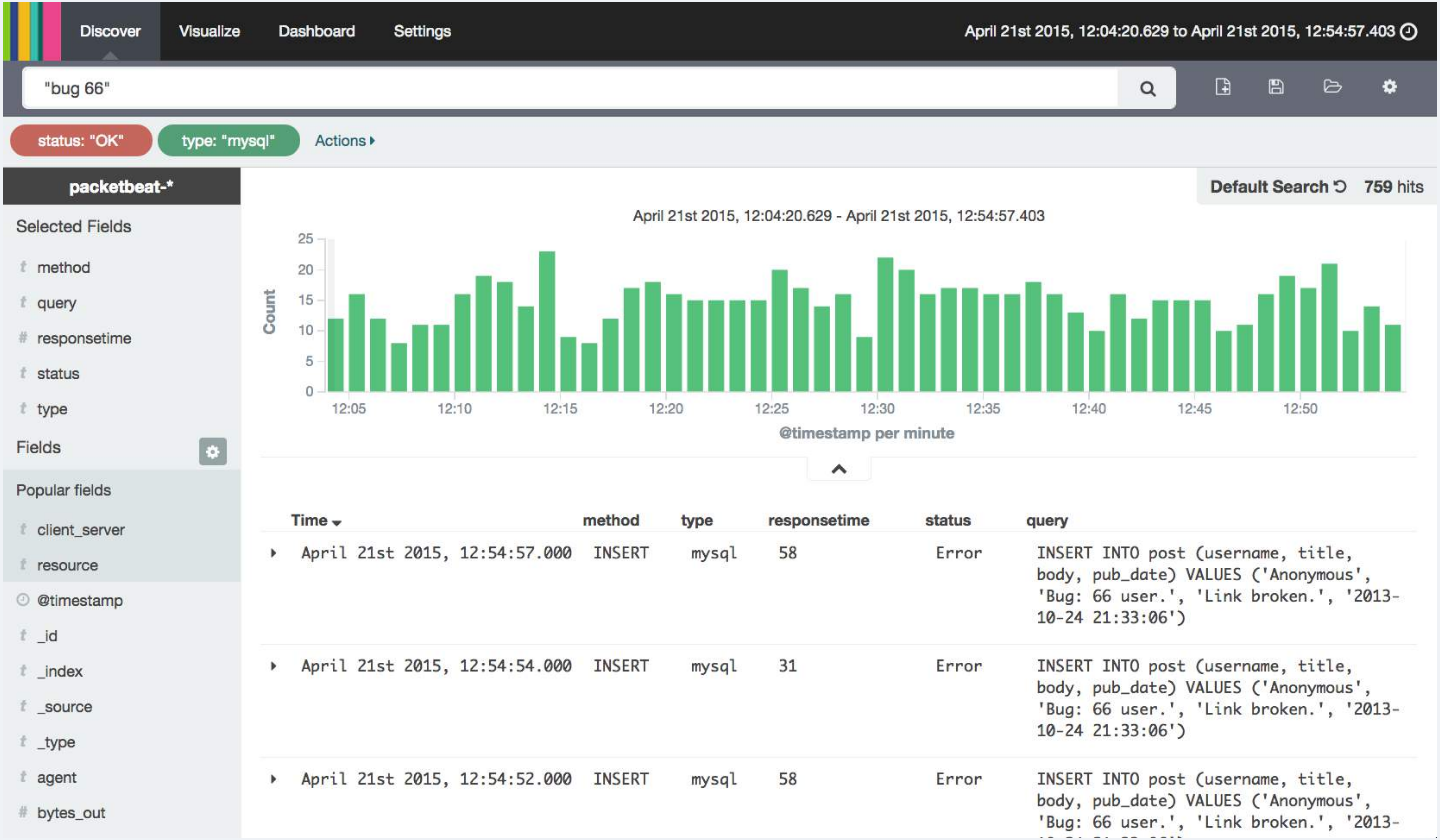
Packetbeat + ELK



Why ELK?

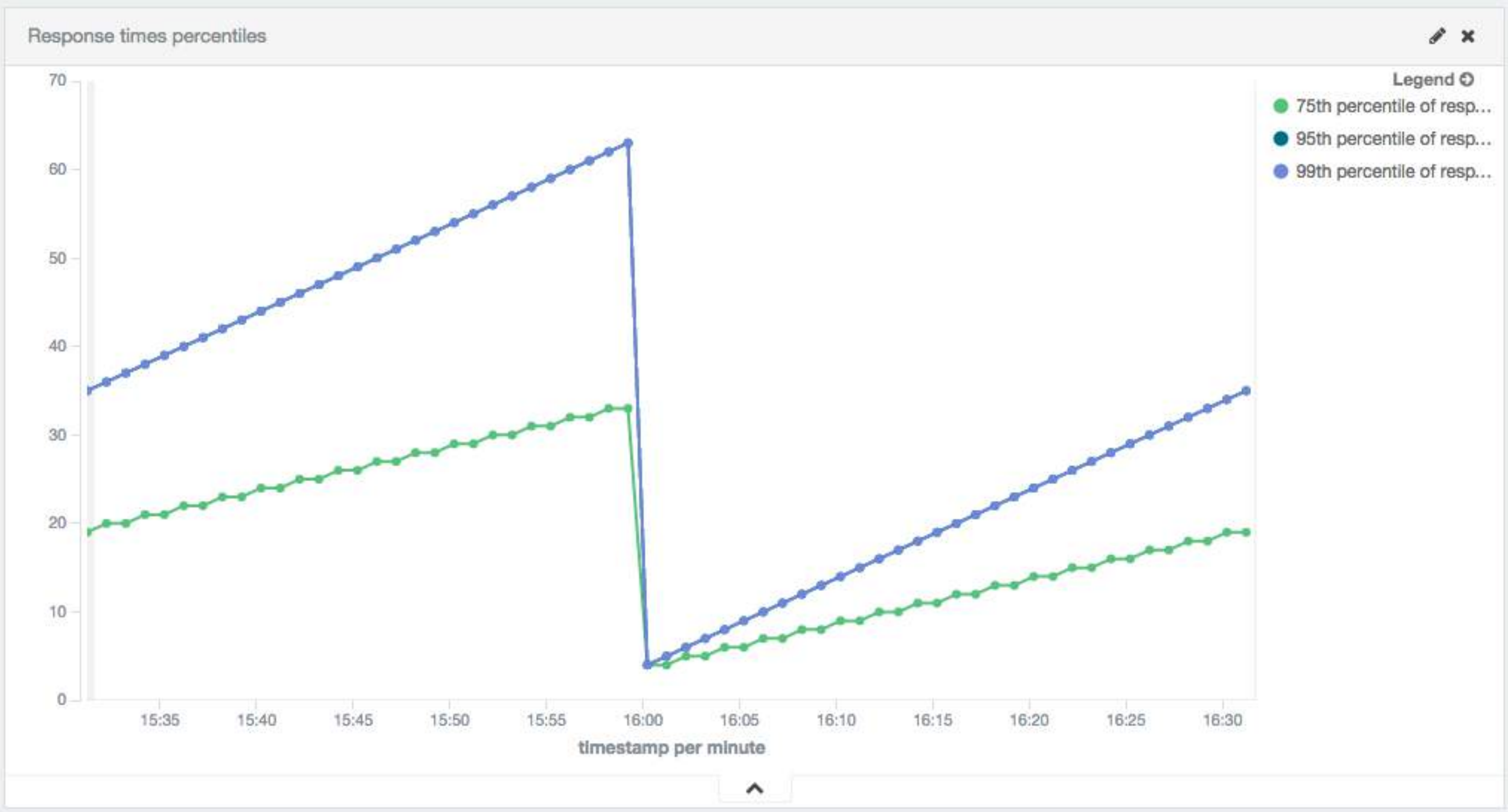
- Already proven to scale and perform for logs
- Clear and simple flow for the data
- Don’t have to create the metrics beforehand
- Powerful features that become simple:
  - Drilling down to the transactions related to a peak
  - Top N features are trivial
  - Slicing by different dimensions is easy

visualizing the data


Discover
Visualize
Dashboard
Settings
April 21st 2015, 12:04:20.629 to April 21st 2015, 12:54:57.403
"bug 66"
status: "OK"
type: "mysql"
Actions
packetbeat-*
Default Search
759 hits
Selected Fields
method
query
responsetime
status
type
Fields
Popular fields
client_server
resource
@timestamp
_id
_index
_source
_type
agent
bytes_out
April 21st 2015, 12:04:20.629 - April 21st 2015, 12:54:57.403
Count
@timestamp per minute
Time
method
type
responsetime
status
query
April 21st 2015, 12:54:57.000 INSERT mysql 58 Error INSERT INTO post (username, title, body, pub_date) VALUES ('Anonymous', 'Bug: 66 user.', 'Link broken.', '2013-10-24 21:33:06')
April 21st 2015, 12:54:54.000 INSERT mysql 31 Error INSERT INTO post (username, title, body, pub_date) VALUES ('Anonymous', 'Bug: 66 user.', 'Link broken.', '2013-10-24 21:33:06')
April 21st 2015, 12:54:52.000 INSERT mysql 58 Error INSERT INTO post (username, title, body, pub_date) VALUES ('Anonymous', 'Bug: 66 user.', 'Link broken.', '2013-

Web transactions
Sum of count
400
300
200
100
0
11:10 11:15 11:20 11:25 11:30 11:35 11:40 11:45 11:50 11:55 12:00
timestamp per minute
DB transactions
Sum of count
2.5k
2k
1.5k
1k
500
0
11:10 11:20 11:30 11:40 11:50 12:00
timestamp per minute
Legend
mysql
pgsql
RPC transactions
Sum of count
2k
1.5k
1k
500
0
11:10 11:15 11:20 11:25 11:30 11:35 11:40 11:45 11:50 11:55 12:00
timestamp per minute
Cache transactions
Sum of count
2k
1.5k
1k
500
0
11:10 11:15 11:20 11:25 11:30 11:35 11:40 11:45 11:50 11:55 12:00
timestamp per minute
Response times repartition
Sum of count
6k
5k
4k
3k
2k
1k
0
11:05 11:10 11:15 11:20 11:25 11:30 11:35 11:40 11:45 11:50 11:55 12:00
timestamp per minute
Legend
0
10
20
30
40
50
60
Response times percentiles
60
40
20
0
11:05 11:10 11:15 11:20 11:25 11:30 11:35 11:40 11:45 11:50 11:55 12:00
timestamp per minute
Legend
75th percentile of resp...
95th percentile of resp...
99.5th percentile of re...
Latency histogram
Sum of count
150k
100k
50k
0
0 10 20 30 40 50 60
responsetime

Percentile values over time



- Combines **date histogram** and **percentiles** aggregations

Percentiles aggregation

- 95th percentile means that 95% of the values are smaller it

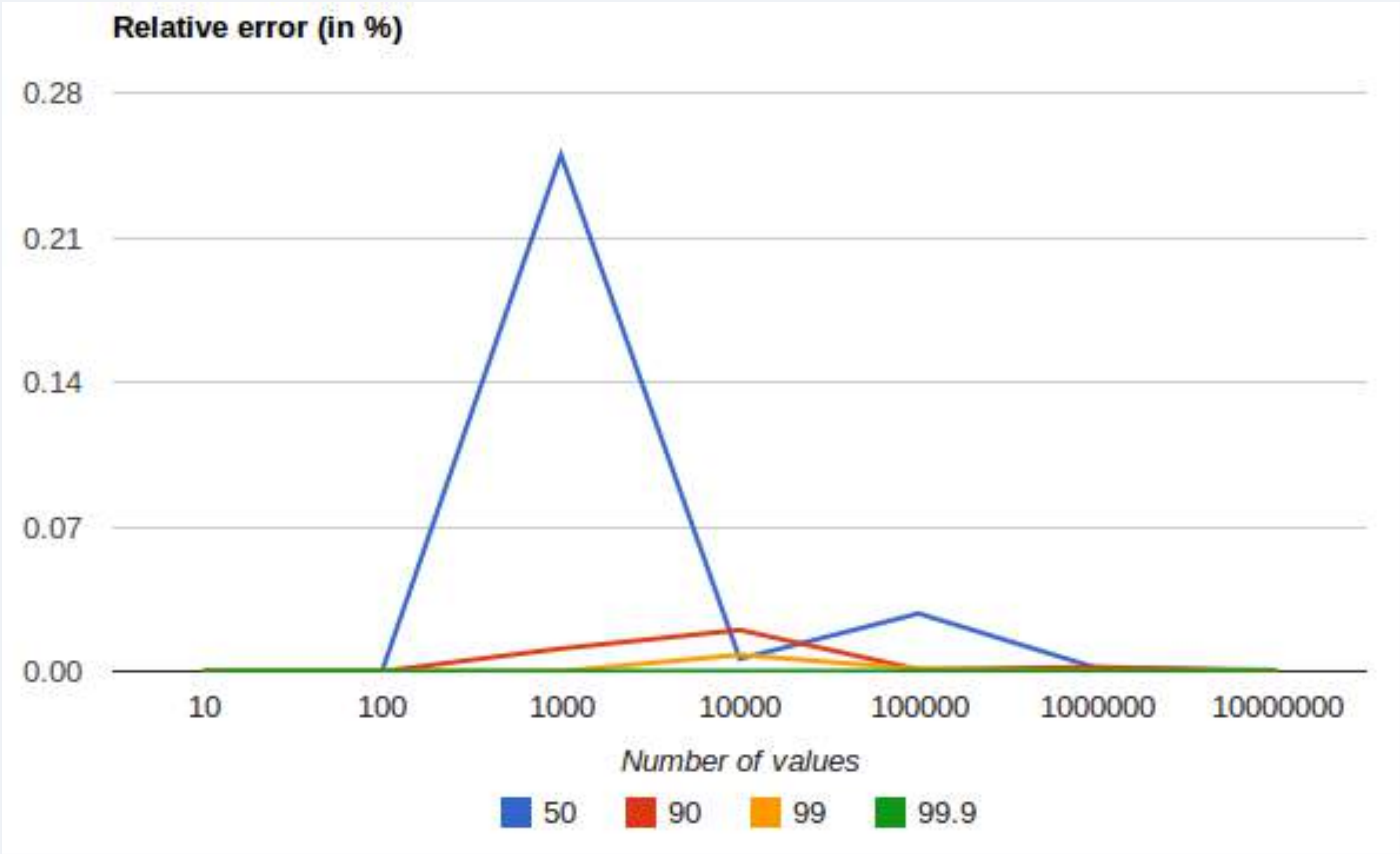
```
"aggs": {
    "responsetime_percentiles": {
        "percentiles": {
            "field": "responsetime",
            "percents": [75, 95, 99.5]
        }
    }
}
```

Response


```
"aggregations": {
    "resoponsetimes_percentiles": {
        "values": {
            "75.0": 21,
            "95.0": 48.72185582951542,
            "99.5": 62
        }
    }
}
```

Percentiles aggregation

- Approximate values
- T-digests algorithm by Ted Dunning
- Accurate for small sets of values
- More accurate for extreme percentiles



Date histogram

- Splits data in buckets of time
- Example:

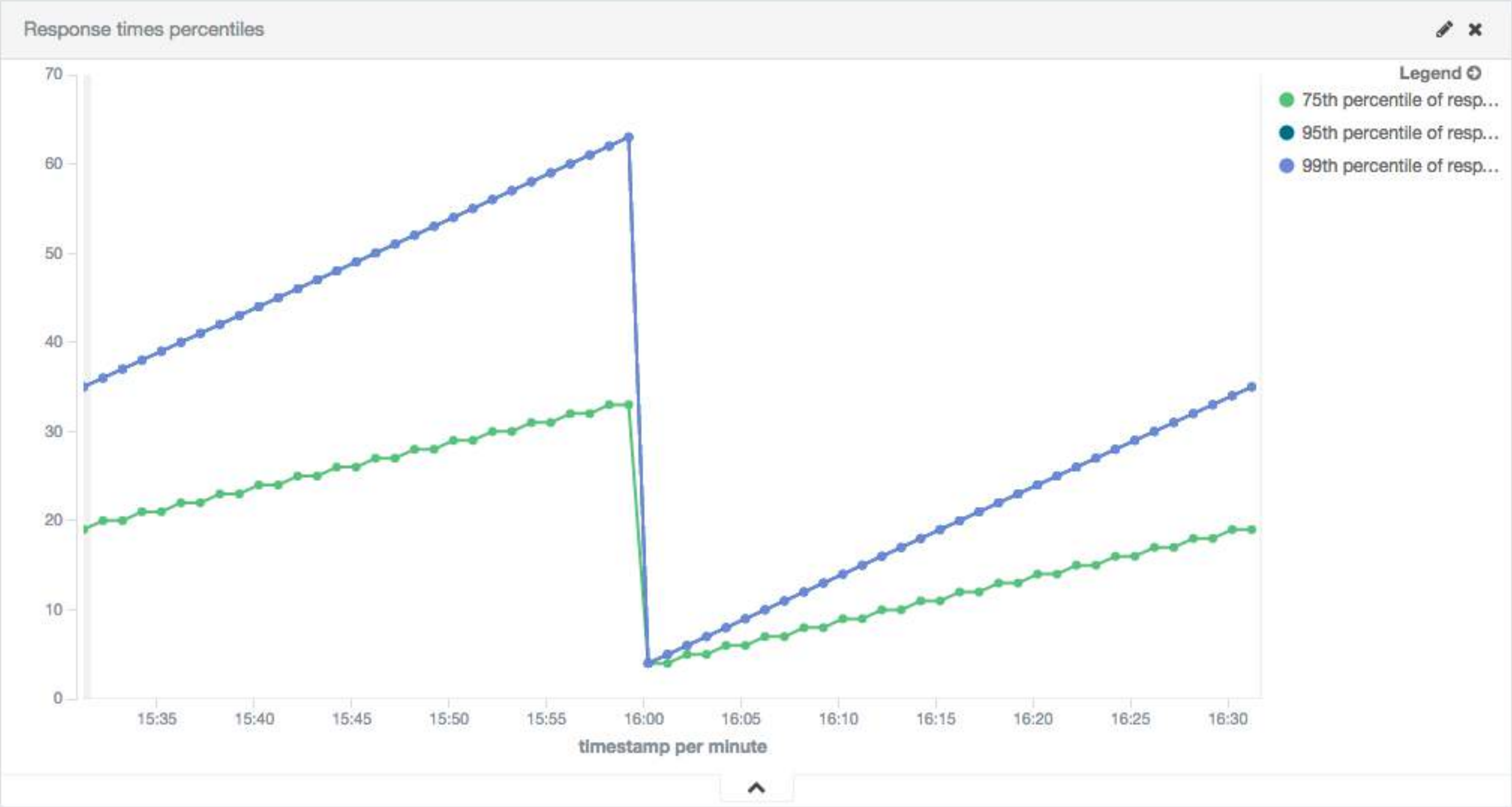
```
"aggs": {
    "per_1m": {
        "date_histogram": {
            "field": "timestamp",
            "interval": "1m"
        }
    }
}
```

```
"aggregations": {
  "per_1m": {
    "buckets": [
      {
        "key_as_string": "2015-05-30T14:21:00.000Z",
        "key": 1432995660000,
        "doc_count": 1425
      },
      {
        "key_as_string": "2015-05-30T14:22:00.000Z",
        "key": 1432995720000,
        "doc_count": 5726
      },
      {
        "key_as_string": "2015-05-30T14:23:00.000Z",
        "key": 1432995780000,
```

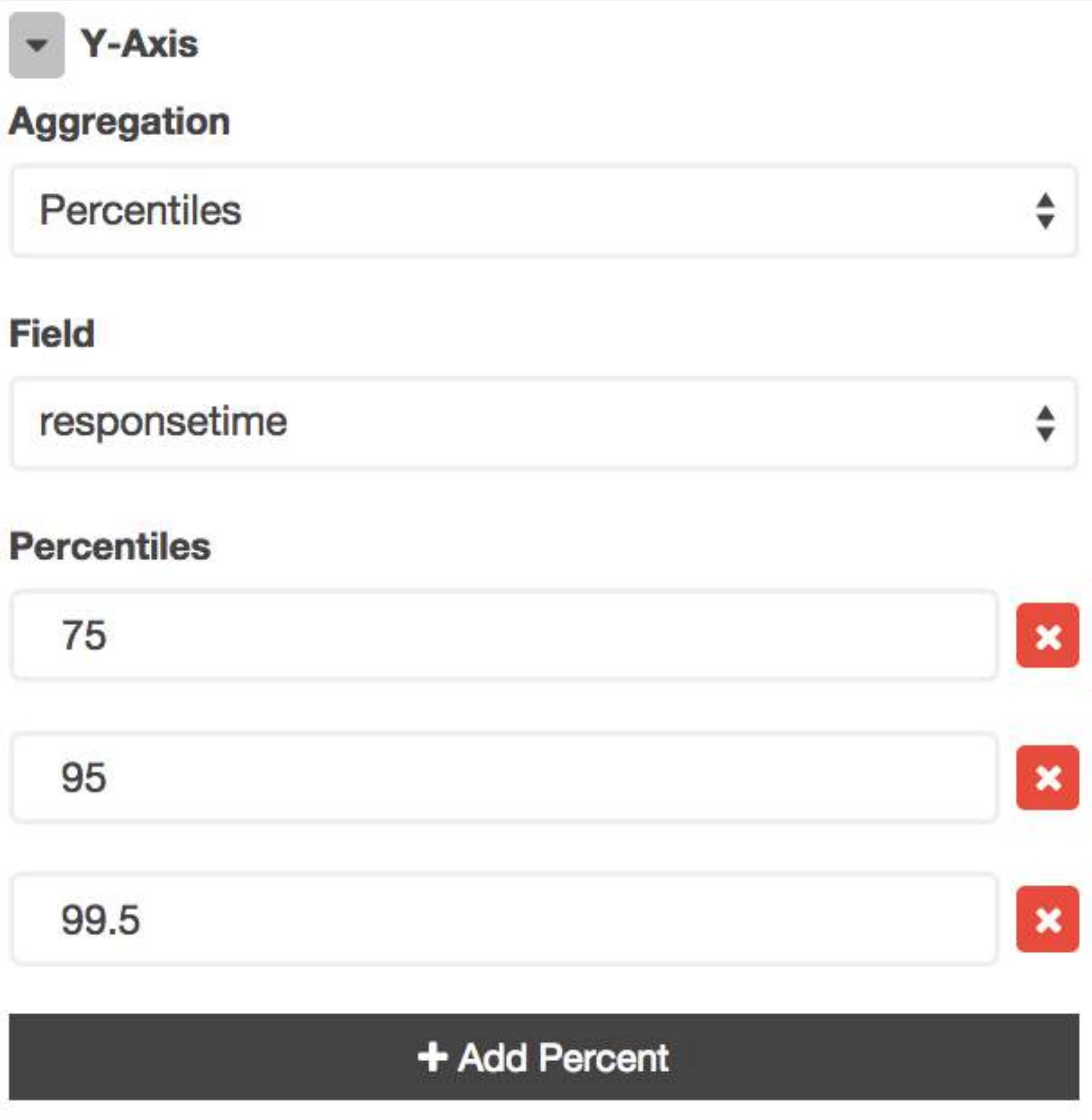
Date histogram nested with percentiles

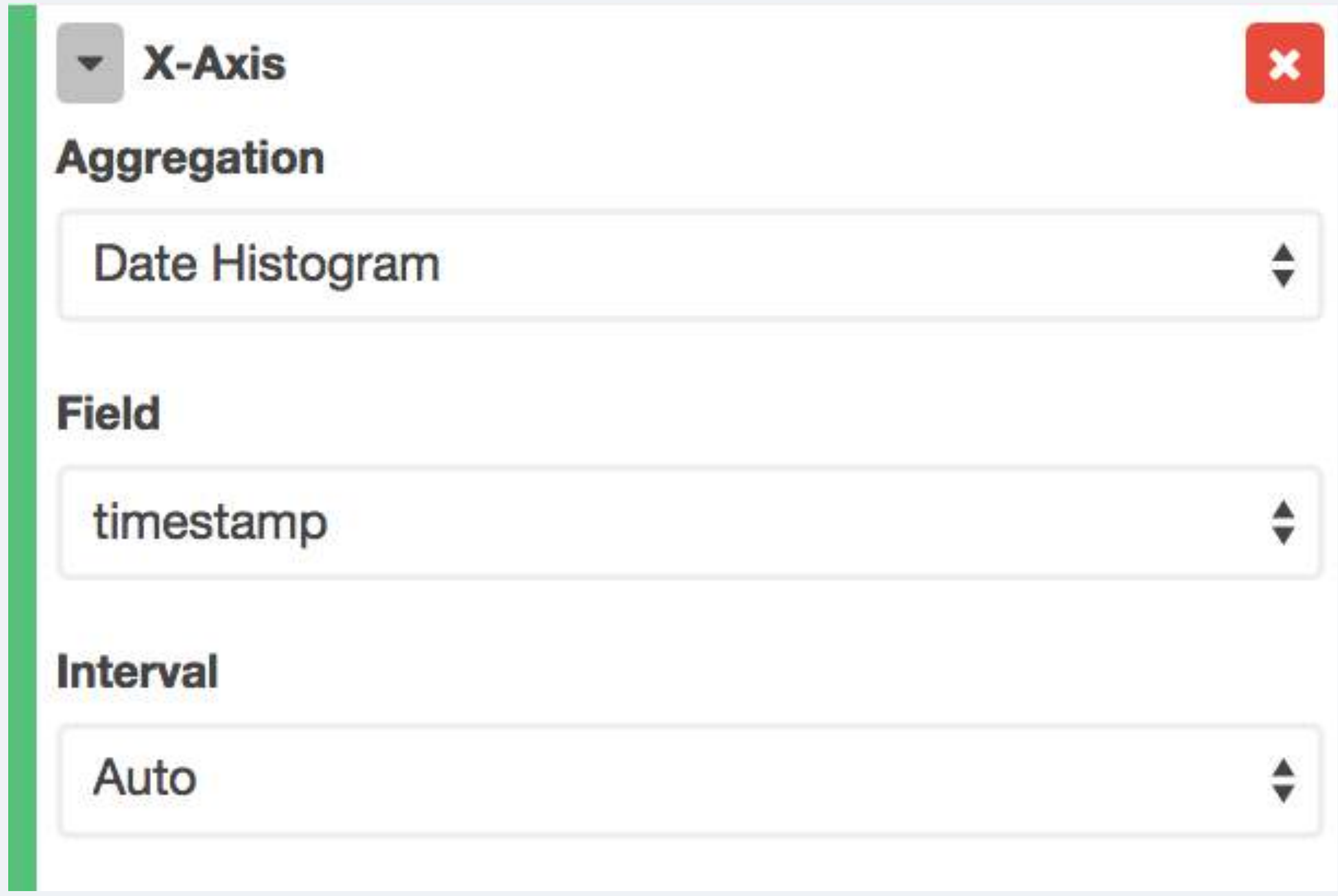
```
"aggs": {
  "per_1m": {
    "date_histogram": { "field": "timestamp", "interval": "1m" },
    "aggs": {
      "responsetime_percentiles": {
        "percentiles": {
          "field": "responsetime",
          "percents": [75, 95, 99.5]
        }
      }
    }
  }
}
```

```
"aggregations": {
  "per_1m": {
    "buckets": [
      {
        "key_as_string": "2015-05-30T15:03:00.000Z",
        "key": 1432998180000,
        "doc_count": 1805,
        "responsetime_percentiles": {
          "values": {
            "75.0": 5,
            "95.0": 7,
            "99.5": 7
          }
        }
      },
      {
        "key_as_string": "2015-05-30T15:04:00.000Z",
        "key": 1432998240000,
        "doc_count": 5719,
        "responsetime_percentiles": {
          "values": {
            "75.0": 6,
            "95.0": 8,
            "99.5": 8
          }
```

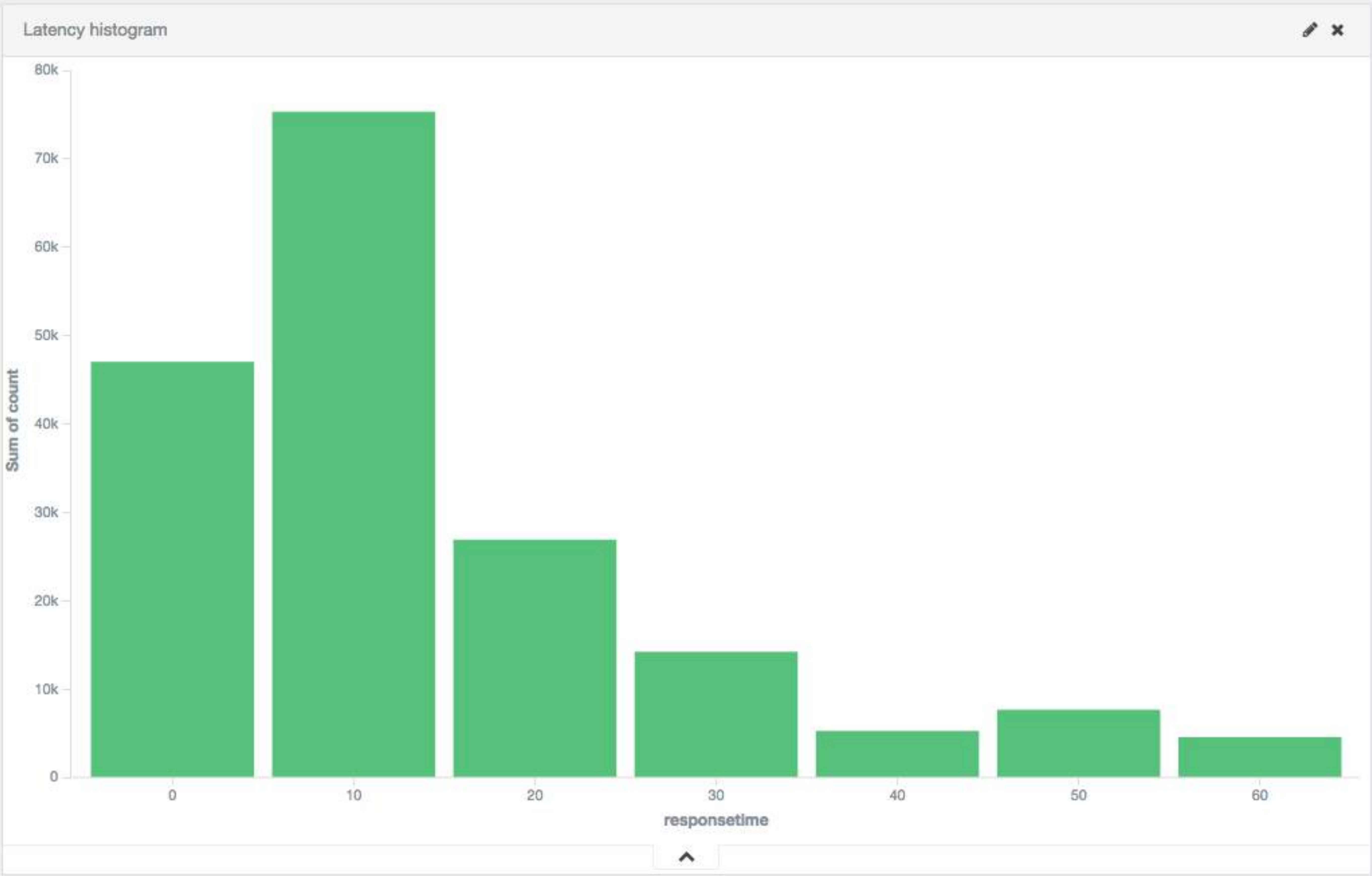

Response times percentiles
70
60
50
40
30
20
10
0
15:35
15:40
15:45
15:50
15:55
16:00
16:05
16:10
16:15
16:20
16:25
16:30
timestamp per minute
Legend
75th percentile of resp...
95th percentile of resp...
99th percentile of resp...

Kibana config





Latency histogram

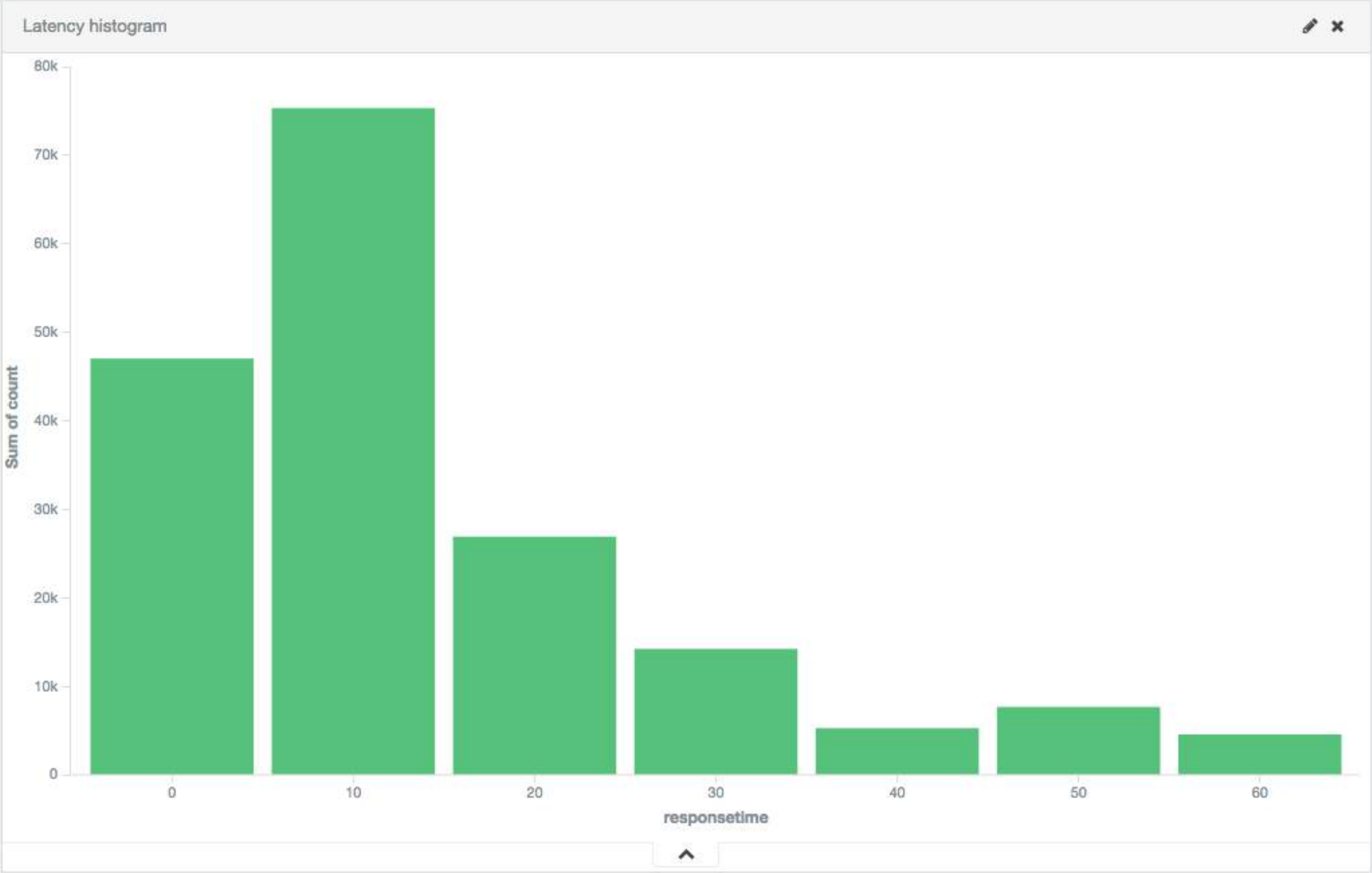


Histogram by response time

- Splits data in buckets by response time
- [0-10ms), [10ms-20ms), …

```
"aggs": {
    "responsetime_histogram": {
        "histogram": {
            "field": "responsetime",
            "interval": 10
        }
    }
}
```

```
"aggregations": {
  "responsetime_histogram": {
    "buckets": [
      {
        "key": 0,
        "doc_count": 47033
      },
      {
        "key": 10,
        "doc_count": 86225
      },
      {
        "key": 20,
        "doc_count": 33801
      },
      {
        "key": 30,
        "doc_count": 14424
```

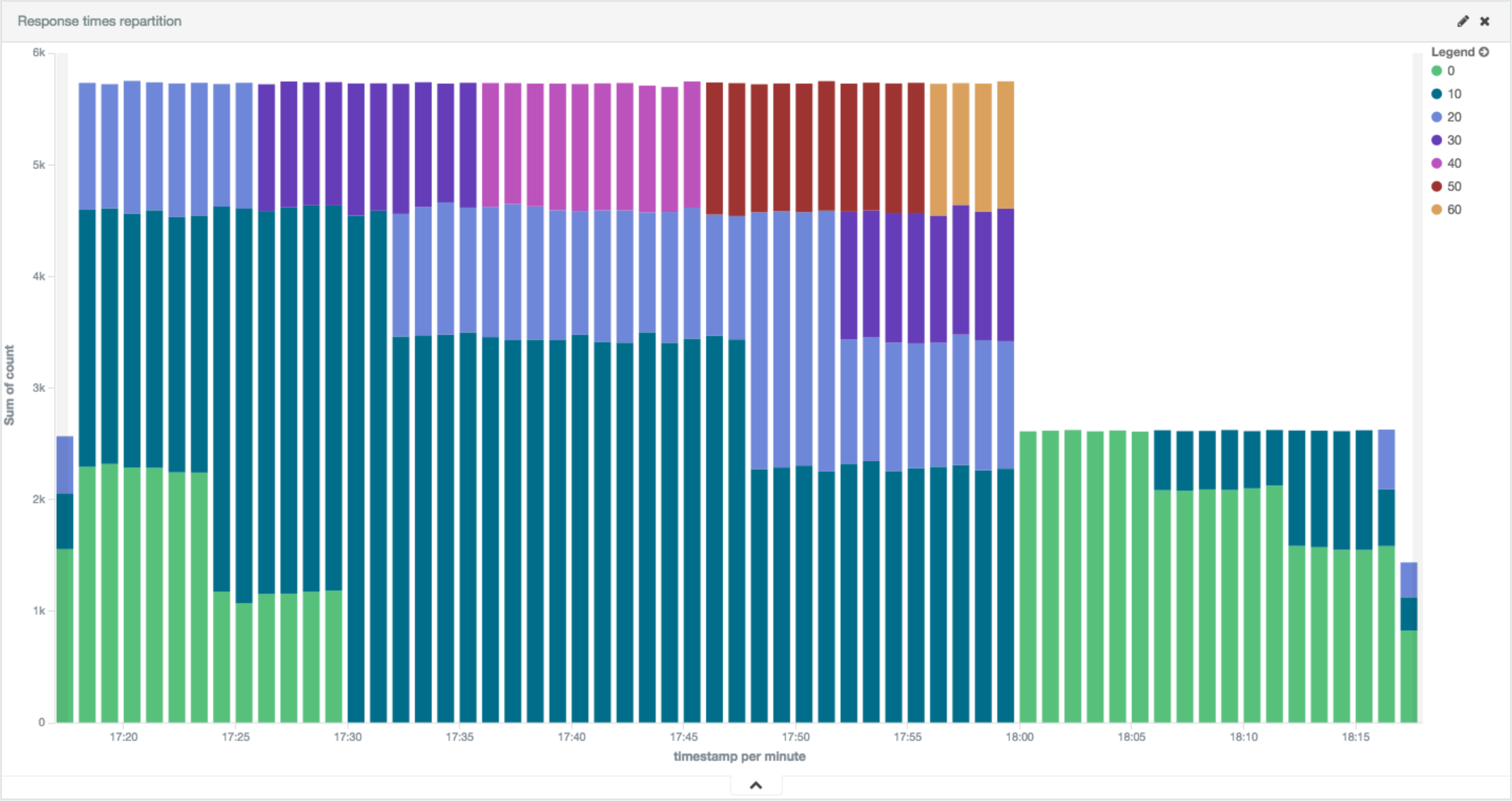

Latency histogram
80k
70k
60k
50k
40k
30k
20k
10k
0
Sum of count
0
10
20
30
40
50
60
responsetime

Add a date histogram

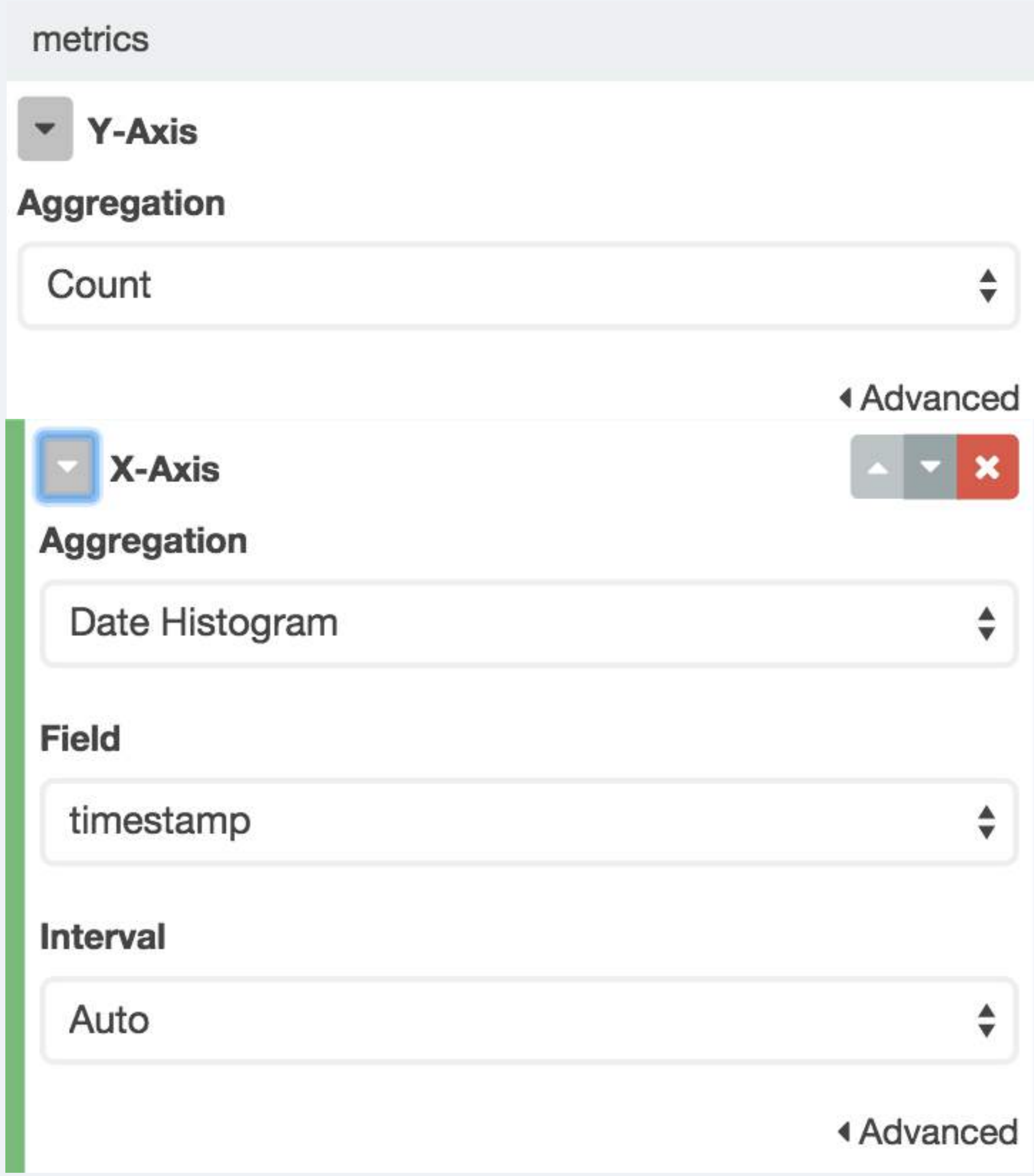
```
"aggs": {
  "per_1m": {
    "date_histogram": { "field": "timestamp", "interval": "1m" },
    "aggs": {
      "responsetime_histogram": {
        "histogram": {
          "field": "responsetime",
          "interval": 10
        }
      }
    }
  }
}
```

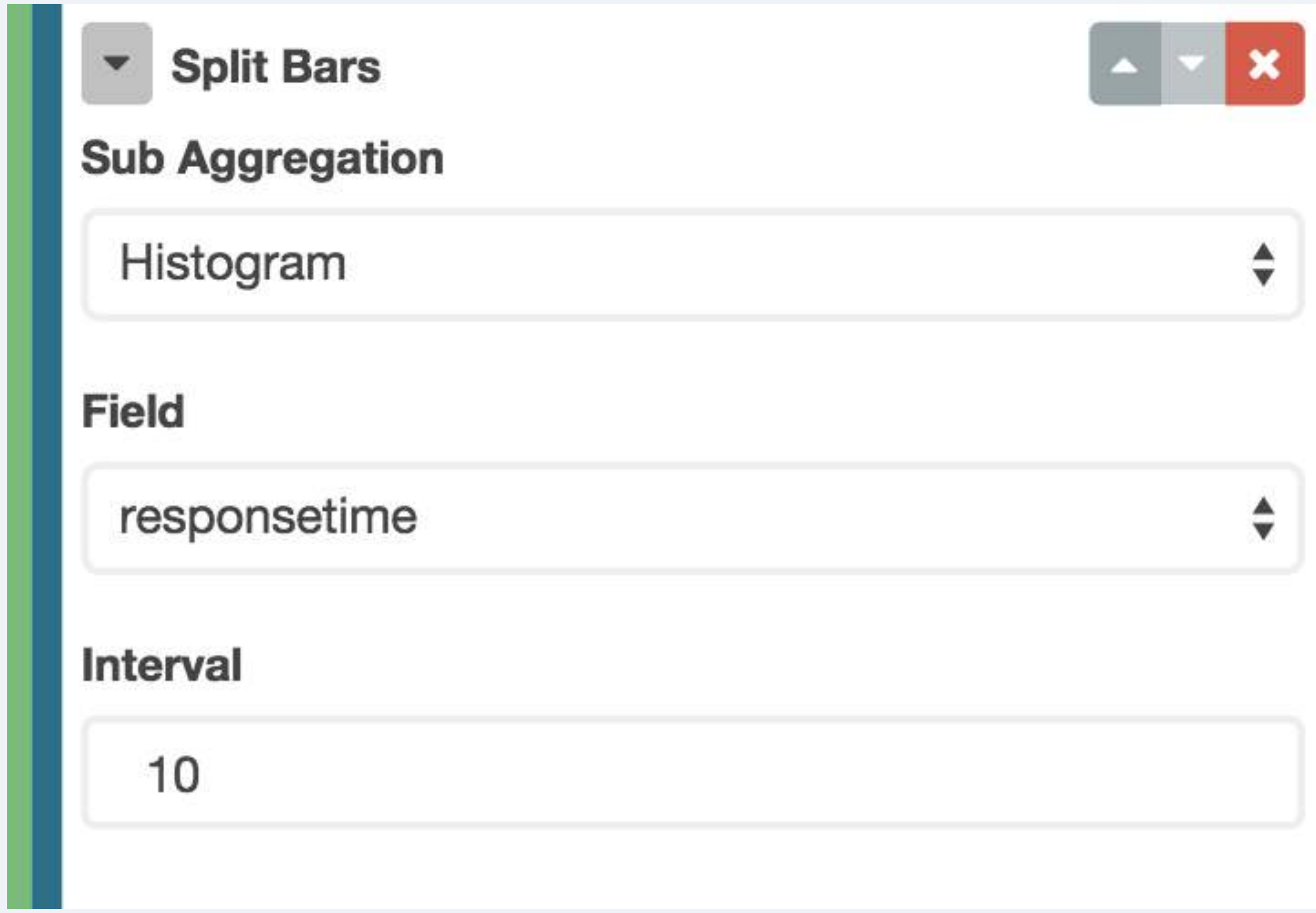
```
"aggregations": {
  "per_1m": {
    "buckets": [
      {
        "key_as_string": "2015-05-31T15:59:00.000Z",
        "key": 1433087940000,
        "doc_count": 1918,
        "responsetime_histogram": {
          "buckets": [
            {
              "key": 10,
              "doc_count": 771
            },
            {
              "key": 20,
              "doc_count": 373
            },
            {
              "key": 30,
              "doc_count": 396
            },
            {
              "key": 60,
              "doc_count": 378
            }
          ]
        }
      },
      {
        "key_as_string": "2015-05-31T16:00:00.000Z",
        "key": 1433088000000,
```

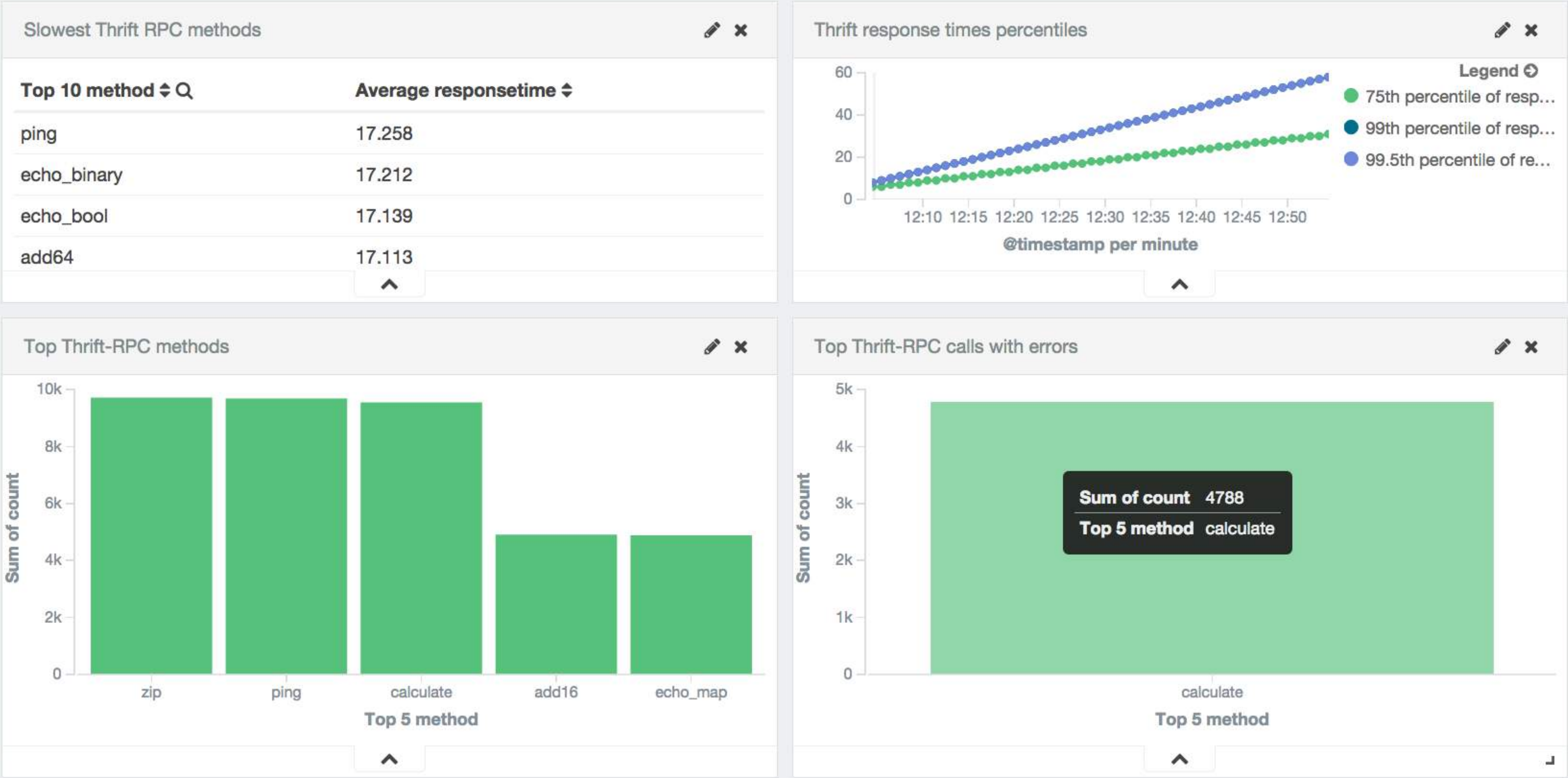
Response times repartition



Kibana config






Slowest Thrift RPC methods
Top 10 method
Average responsetime
ping 17.258
echo_binary 17.212
echo_bool 17.139
add64 17.113
Thrift response times percentiles
Legend
75th percentile of resp...
99th percentile of resp...
99.5th percentile of re...
60
40
20
0
12:10 12:15 12:20 12:25 12:30 12:35 12:40 12:45 12:50
@timestamp per minute
Top Thrift-RPC methods
Sum of count
10k
8k
6k
4k
2k
0
zip
ping
calculate
add16
echo_map
Top 5 method
Top Thrift-RPC calls with errors
Sum of count
5k
4k
3k
2k
1k
0
Sum of count 4788
Top 5 method calculate
calculate
Top 5 method

Slowest RPC methods

Slowest Thrift RPC methods

Top 10 method	99th percentile of responsetime
add64	62
add	61
add16	61
echo_map	61
echo_set	61
getStruct	61
zip	61
echo_bool	60.19
calculate	60
echo_binary	60

Export: Raw Formatted

- Combines **terms** and **percentiles** aggregations

Terms aggregation

- Buckets are dynamically built: one per unique value
- By default: top 10 by document count
- Approximate because each shard can have a different top 10

```
"aggs": {
    "methods": {
        "terms": {
            "field": "method",
            "size": 10
        }
    }
}
```

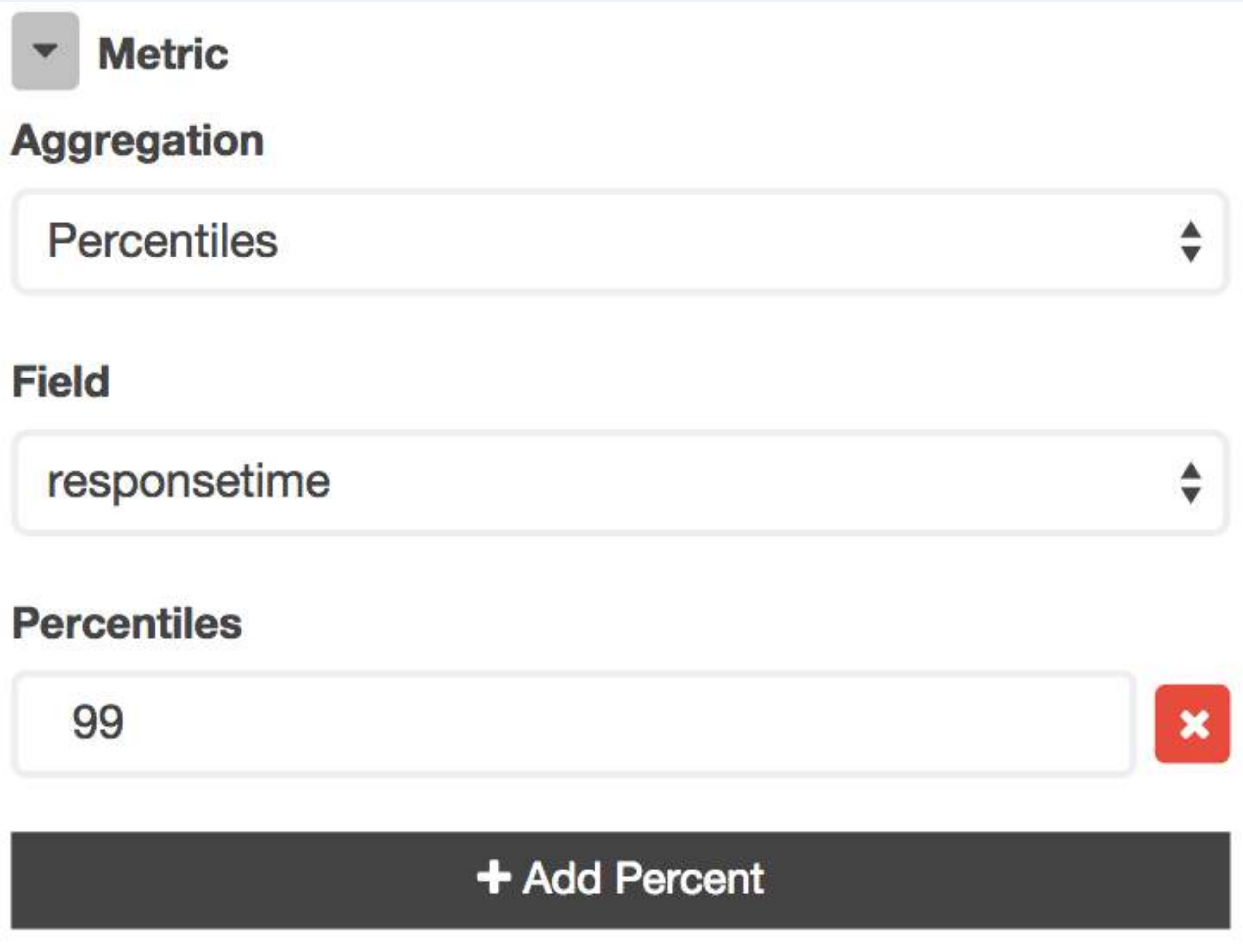
```
"aggregations": {
  "methods": {
    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 6373,
    "buckets": [
      {
        "key": "calculate",
        "doc_count": 3359
      },
      {
        "key": "zip",
        "doc_count": 3331
      },
      {
        "key": "ping",
        "doc_count": 3280
      },
      {
```

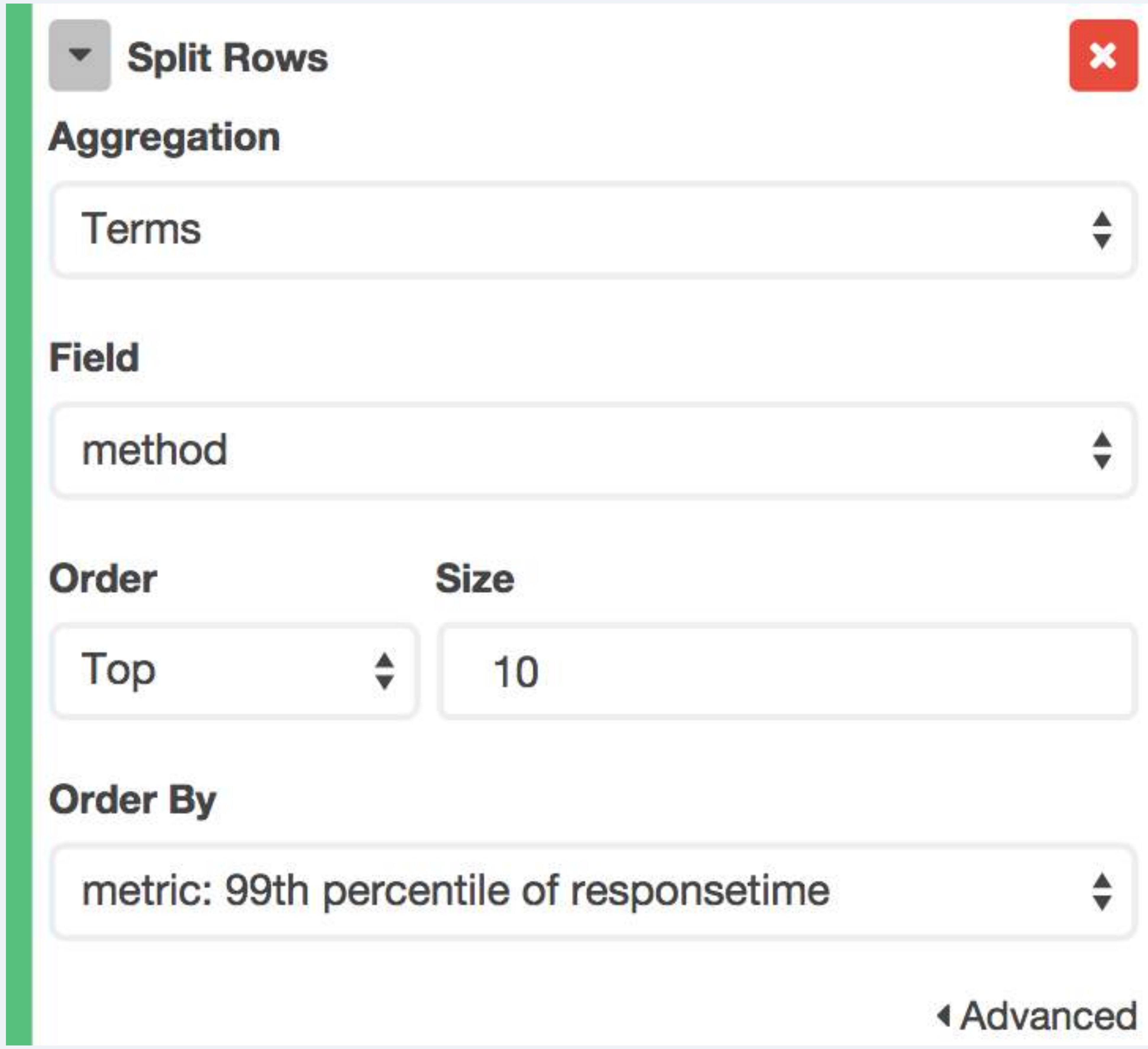
Order by 99th percentile

```
"aggs": {
  "methods": {
    "terms": {
      "field": "method", "size": 10,
      "order": { "responsetimes.99": "desc" }
    },
    "aggs": {
      "responsetimes": {
        "percentiles": {
          "field": "responsetime",
          "percents": [99]
        }
      }
    }
  }
}
```

```
"aggregations": {
  "methods": {
    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 8813,
    "buckets": [
      {
        "key": "add64",
        "doc_count": 1735,
        "responsetimes": {
          "values": {
            "99.0": 62
          }
        }
      },
      {
        "key": "echo_list",
        "doc_count": 1679,
        "responsetimes": {
          "values": {
            "99.0": 62
          }
        }
      },
      {
        "key": "echo_string",
```

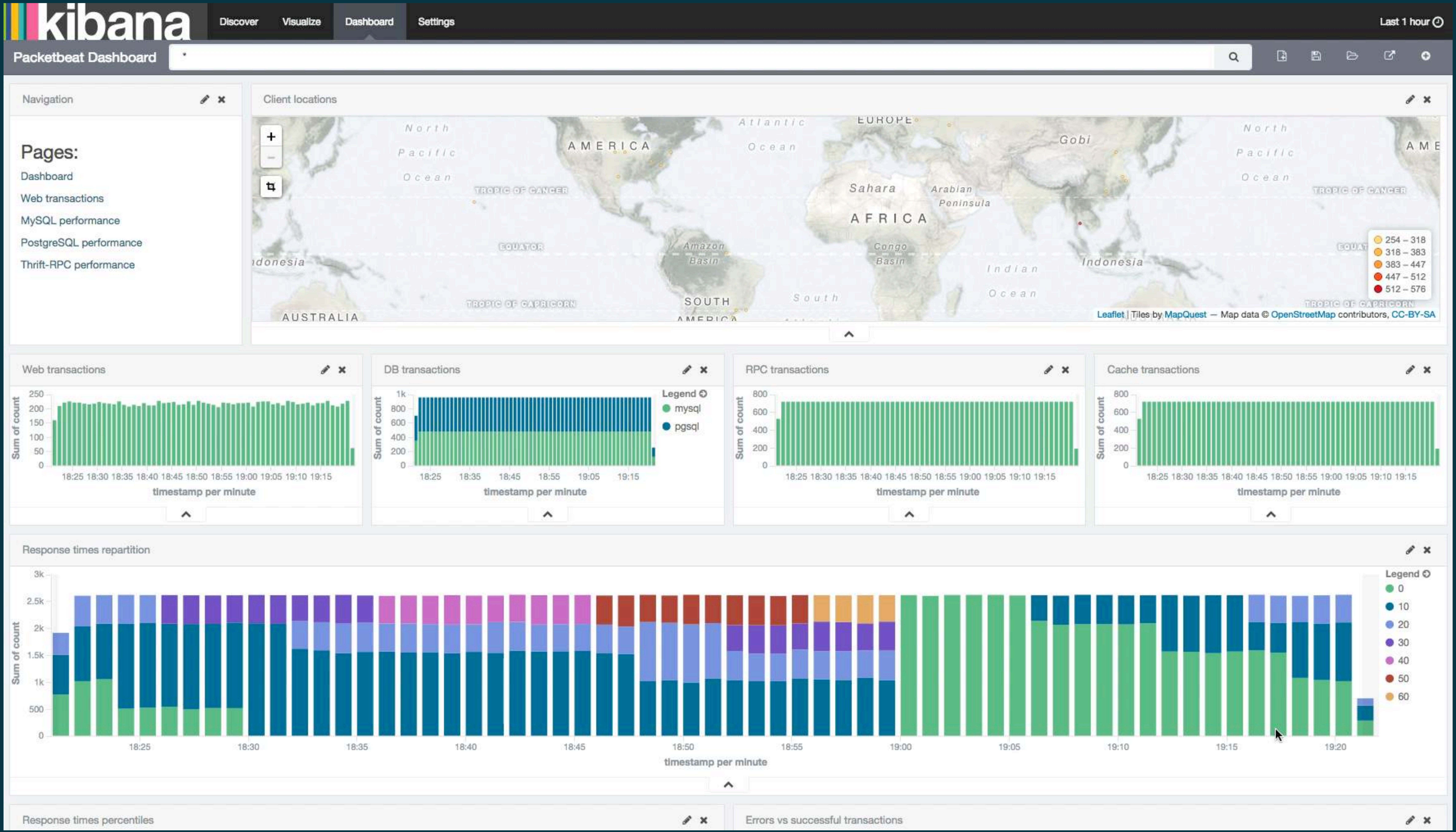
Kibana config





Tips

- Live demo: http://demo.elastic.co/packetbeat/
- All examples here: https://github.com/tsg/bbuzz2015
- Use Sense (chrome add-on)

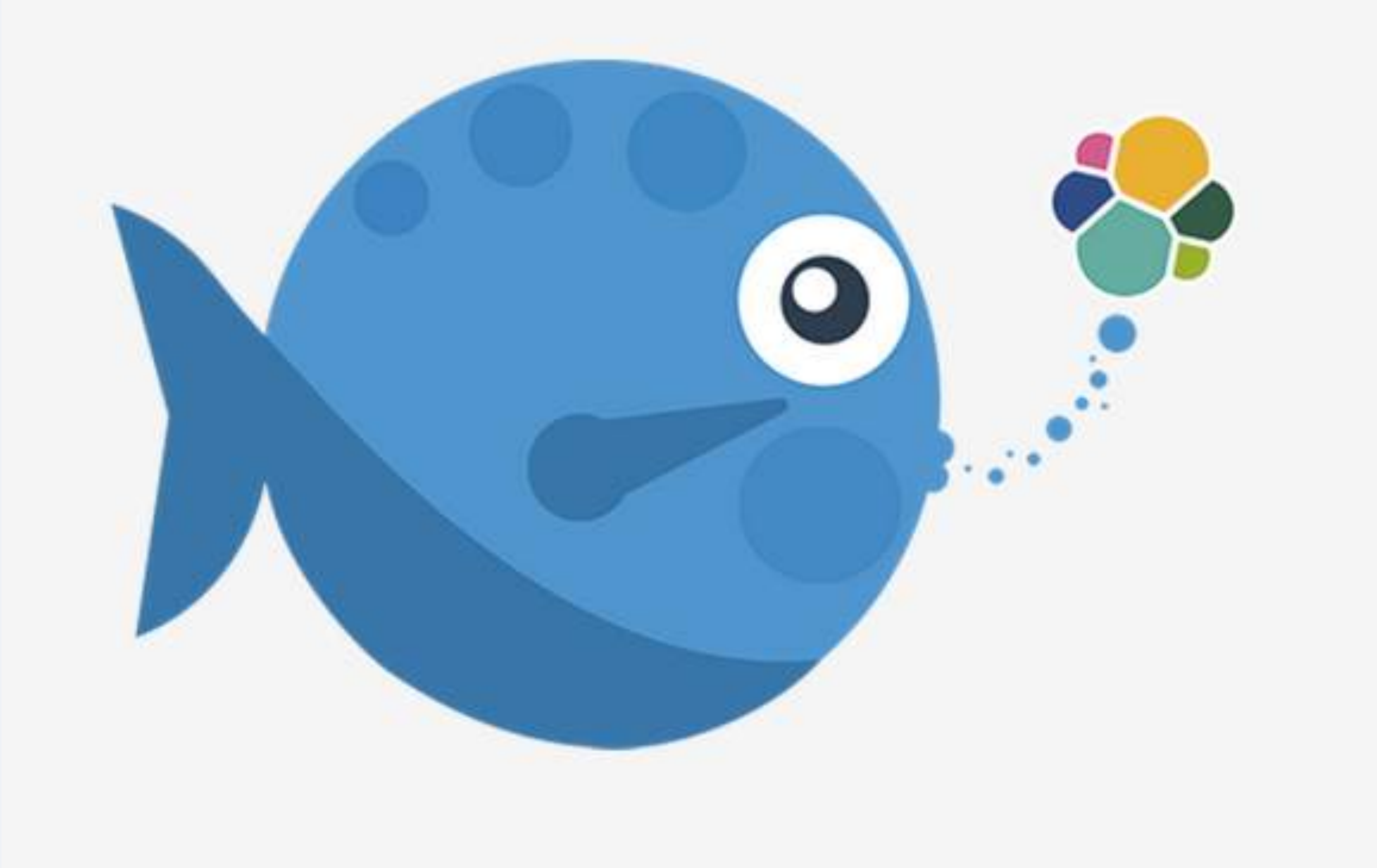

kibana
Discover
Visualize
Dashboard
Settings
Last 1 hour
Packetbeat Dashboard
Navigation
Pages:
Dashboard
Web transactions
MySQL performance
PostgreSQL performance
Thrift-RPC performance
Client locations
254 – 318
318 – 383
383 – 447
447 – 512
512 – 576
Leaflet | Tiles by MapQuest — Map data © OpenStreetMap contributors, CC-BY-SA
Web transactions
DB transactions
Legend
mysql
pgsql
RPC transactions
Cache transactions
Sum of count
timestamp per minute
Response times repartition
Legend
0
10
20
30
40
50
60
Response times percentiles
Errors vs successful transactions

```
from __future__ import beats
```

Future plans

- Packet data is just the beginning
- Other sources of operational data:
  - OS readings: CPU, memory, IO stats
  - Code instrumentation, tracing
  - API gateways
  - Common servers internal stats (Nginx, Elasticsearch)

Joining Elastic



ship operational
data to
elasticsearch

The Beats

- Packetbeat - data from the wire
- Filebeat (Logstash-Forwarder) - data from log files
- Future:
  - Topbeat - CPU, mem, IO stats
  - Metricsbeat - arbitrary metrics from nagios/sensu like scripts
  - RUMbeat - data from the browser

Stay in touch

- @packetbeat
- https://discuss.elastic.co/c/beats
- Sign up for the webinar:
  - https://www.elastic.co/webinars/beats-platform-for-leveraging-operational-data