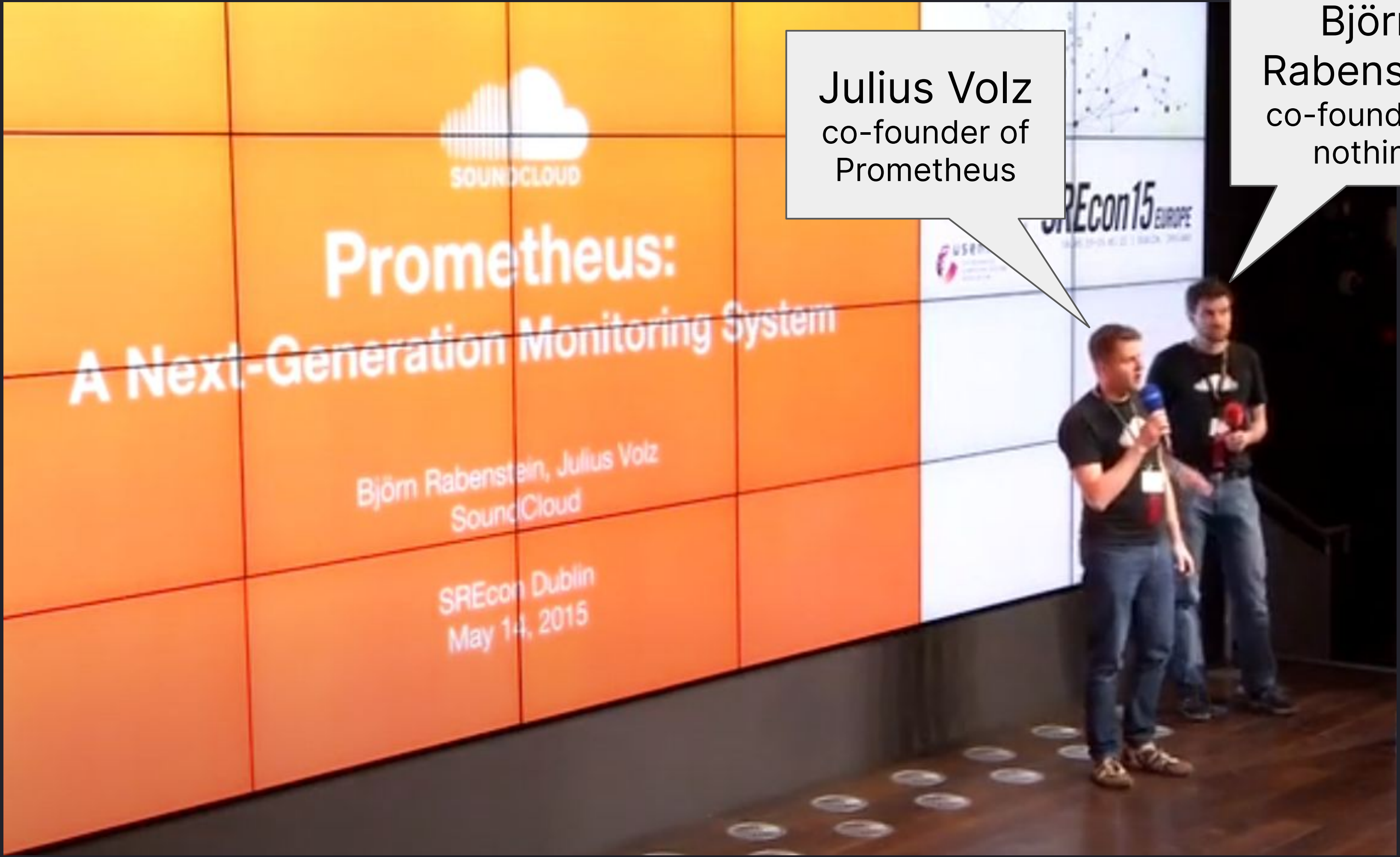
Grafana Labs

How to Use Prometheus's Native Histograms

Björn “Beorn” Rabenstein

SREcon EMEA, Dublin, 2023-10-12.



Julius Volz
co-founder of
Prometheus

Björn
Rabenstein
co-founder of
nothing

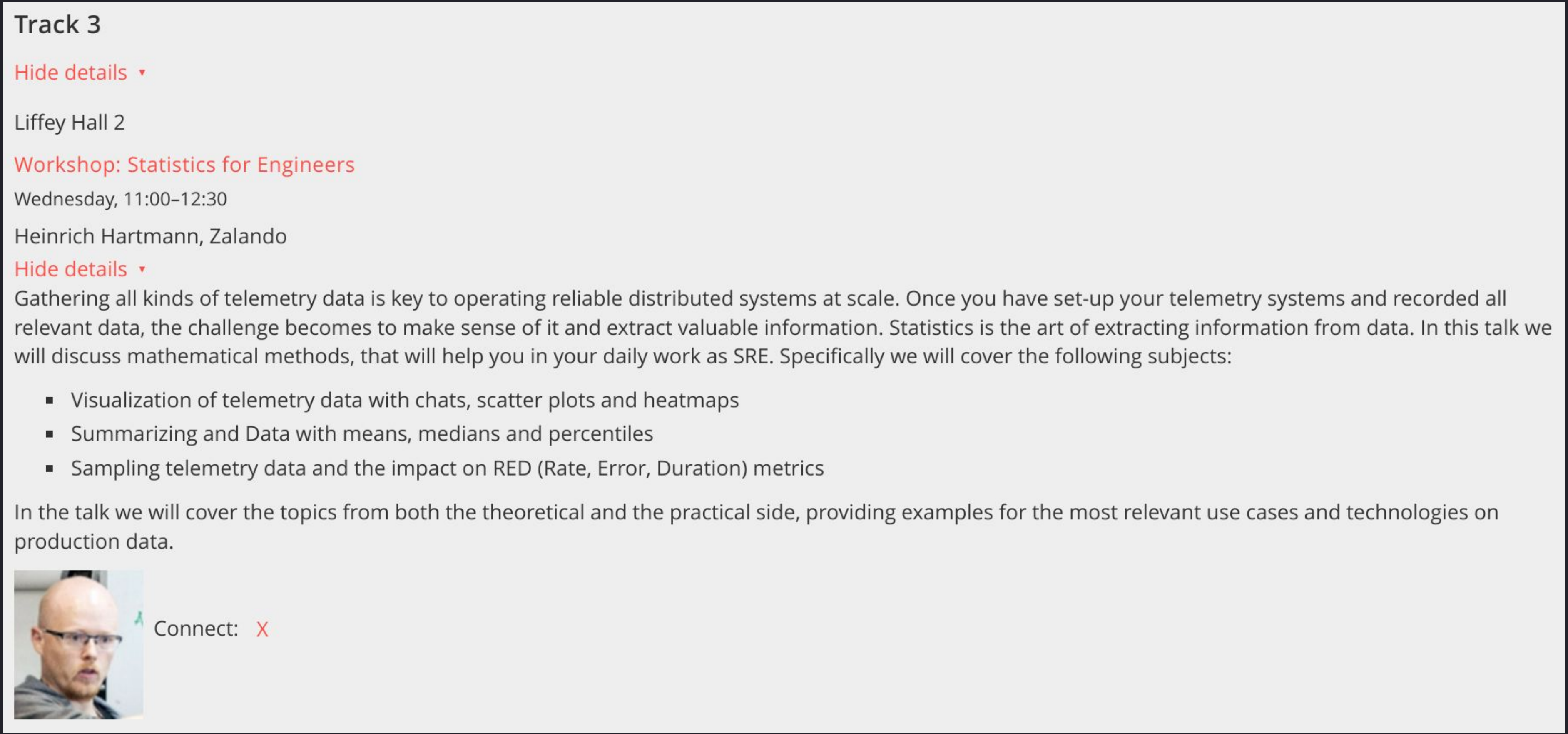
Brendan Burns
co-founder of
Kubernetes

SRE

Prometheus

Histograms

https://www.usenix.org/conference/srecon15europe/program/presentation/rabenstein



Track 3

Hide details ▾

Liffey Hall 2

Workshop: Statistics for Engineers

Wednesday, 11:00–12:30

Heinrich Hartmann, Zalando

Hide details ▾

Gathering all kinds of telemetry data is key to operating reliable distributed systems at scale. Once you have set-up your telemetry systems and recorded all relevant data, the challenge becomes to make sense of it and extract valuable information. Statistics is the art of extracting information from data. In this talk we will discuss mathematical methods, that will help you in your daily work as SRE. Specifically we will cover the following subjects:

- Visualization of telemetry data with chats, scatter plots and heatmaps
- Summarizing and Data with means, medians and percentiles
- Sampling telemetry data and the impact on RED (Rate, Error, Duration) metrics

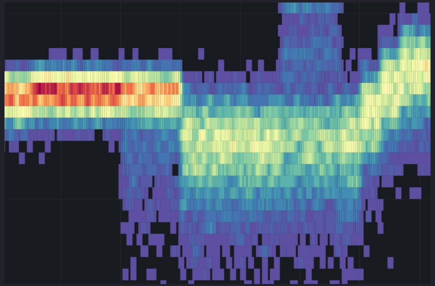
In the talk we will cover the topics from both the theoretical and the practical side, providing examples for the most relevant use cases and technologies on production data.

Connect: X

By Apdex - Apdex Web site, Fair use,
https://en.wikipedia.org/w/index.php?curid=8994240

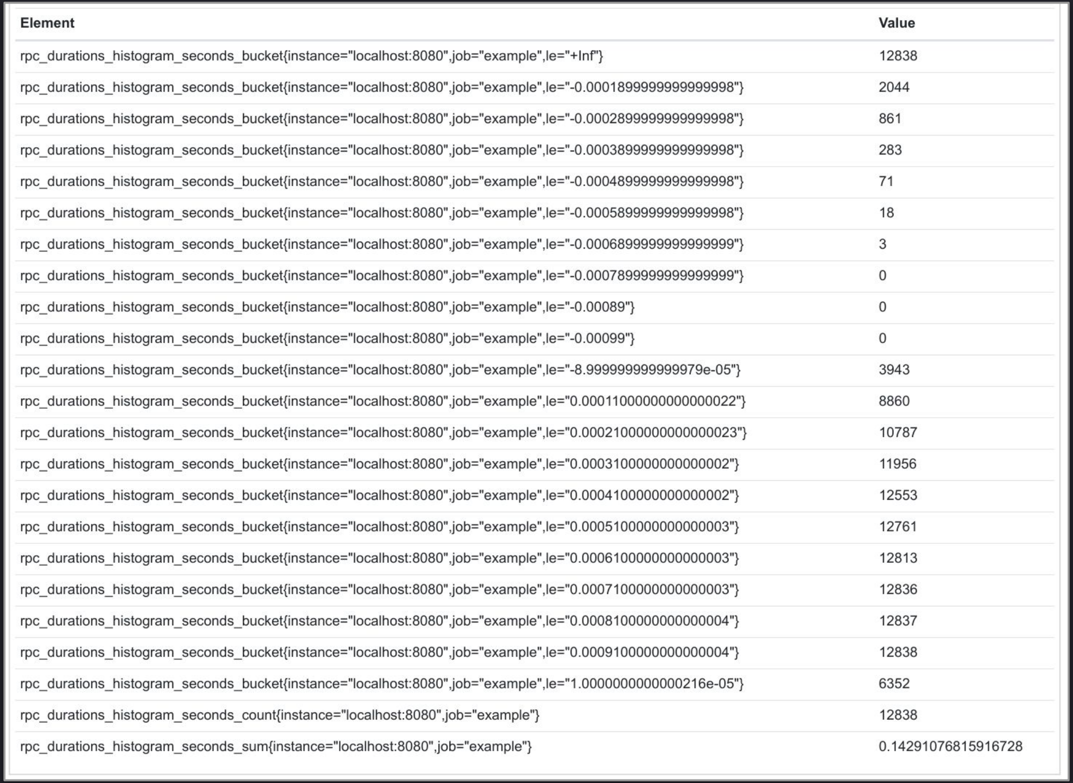
“What percentage of requests in the last hour got a response in 100ms or less?”

SLO tracking



“How many HTTP responses larger than 4kiB were served on 2019-11-03 between 02:30 and 02:45?”

Correctly aggregated quantiles



Element	Value
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="+Inf"}	12838
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.0001899999999999998"}	2044
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.0002899999999999998"}	861
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.0003899999999999998"}	283
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.0004899999999999998"}	71
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.0005899999999999998"}	18
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.0006899999999999999"}	3
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.0007899999999999999"}	0
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00089"}	0
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00099"}	0
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-8.999999999999979e-05"}	3943
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00011000000000000022"}	8860
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00021000000000000023"}	10787
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.0003100000000000002"}	11956
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.0004100000000000002"}	12553
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.0005100000000000003"}	12761
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.0006100000000000003"}	12813
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.0007100000000000003"}	12836
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.0008100000000000004"}	12837
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.0009100000000000004"}	12838
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="1.0000000000000216e-05"}	6352
rpc_durations_histogram_seconds_count{instance="localhost:8080",job="example"}	12838
rpc_durations_histogram_seconds_sum{instance="localhost:8080",job="example"}	0.14291076815916728

- Need to define buckets during instrumentation.
- Changing them is painful, breaks aggregation.
- Buckets are expensive.
- Limited resolution.
- Limited partitioning by labels.

```
httpRequests = prometheus.NewCounterVec(
     prometheus.CounterOpts{
          Name:       "http_requests_total",
          Help:       "HTTP requests partitioned by status code.",
     },
     []string{"status"},
)

httpRequestDurations = prometheus.NewHistogram(prometheus.HistogramOpts{
     Name:    "http_durations_seconds",
     Help:    "HTTP latency distribution.",
     Buckets: []float64{.005, .01, .025, .05, .1, .25, .5, 1, 2.5, 5, 10},
})
```

Where did we come from?

Secret History of Prometheus Histograms

FOSDEM 2020, Brussels, Belgium.

https://fosdem.org/2020/schedule/event/histograms/

What was the plan?

Prometheus Histograms – ~~Past~~, Present, and Future

PromCon 2019, Munich, Germany.

https://promcon.io/2019-munich/talks/prometheus-histograms-past-present-and-future/

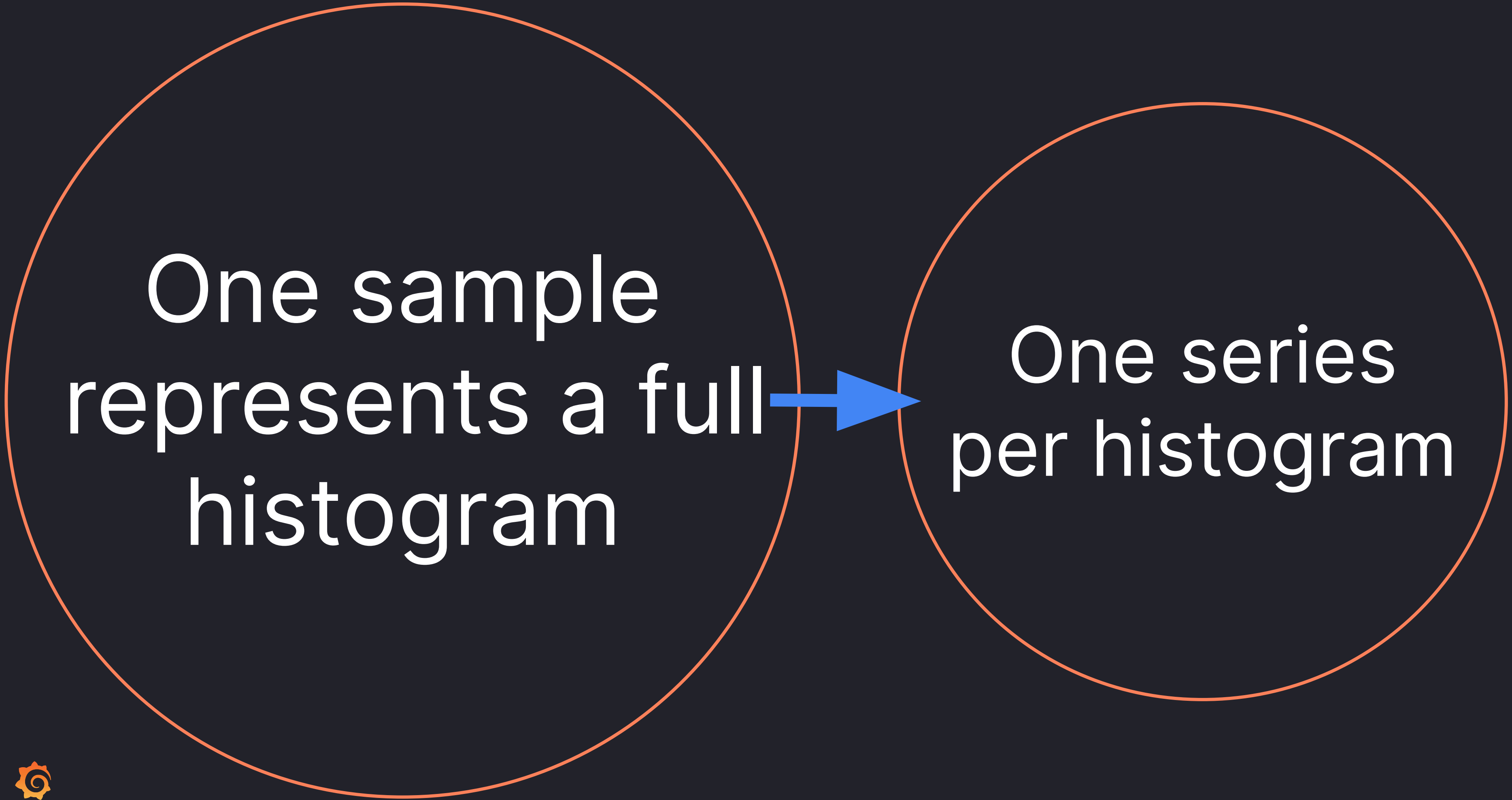
Why does it work?

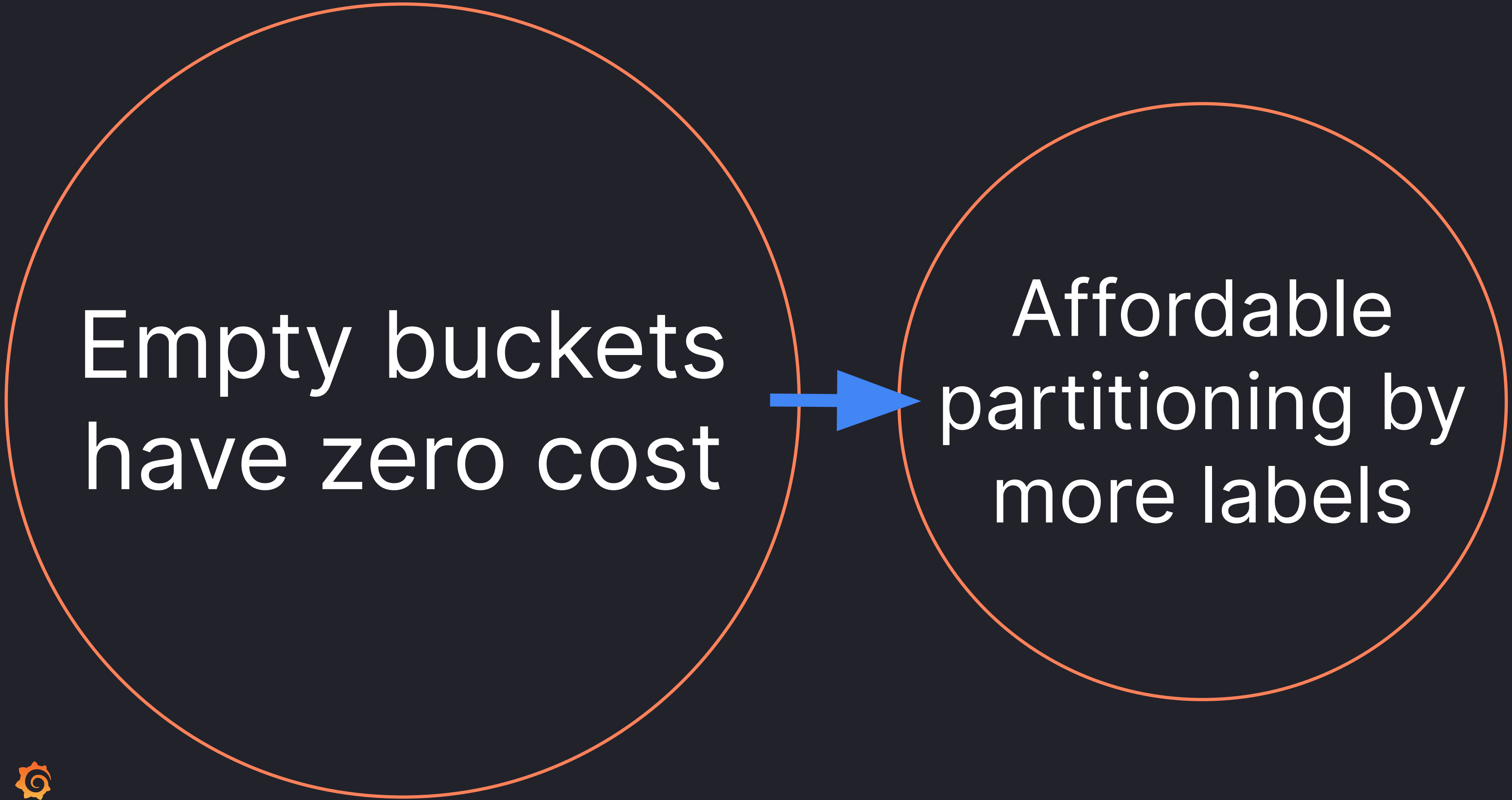
Better Histograms for Prometheus

KubeCon EU 2020, online, anywhere.

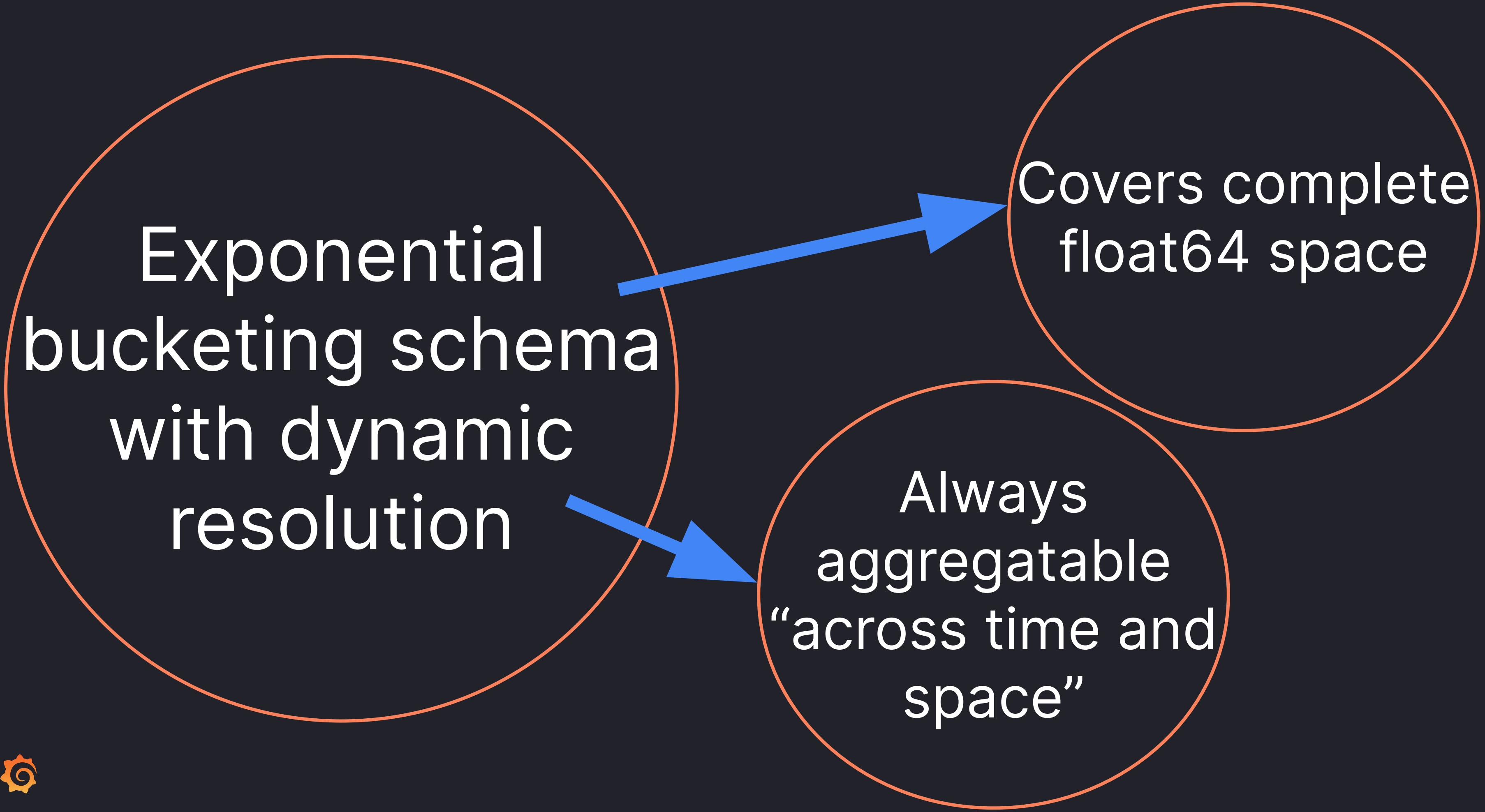
https://www.youtube.com/watch?v=HG7uzON-IDM

Native Histograms


One sample represents a full histogram
One series per histogram


Empty buckets have zero cost
Affordable partitioning by more labels

Minimal configuration during instrumentation


Exponential bucketing schema with dynamic resolution
Covers complete float64 space
Always aggregatable “across time and space”

Affordable high resolution

→

“10x resolution at half the price”

Compatible with

OpenTelemetry

exponential histograms

The fundamentals: (Ganesh Vernekar)

Native Histograms in Prometheus

PromCon 2022, Munich, Germany.

https://promcon.io/2022-munich/talks/native-histograms-in-prometheus/

PromQL changes:

PromQL for Native Histograms

PromCon 2022, Munich, Germany.

https://promcon.io/2022-munich/talks/promql-for-native-histograms/

Performance analysis:

Prometheus Native Histograms in Production

O11y Day (KubeCon) 2023, Amsterdam, Netherlands.

https://www.youtube.com/watch?v=TgINvIK9SYc

OTel compatibility: (Ruslan Kovalov & Ganesh Vernekar)

Using OpenTelemetry's Exponential Histograms in Prometheus

O11y Day (KubeCon) 2023, Amsterdam, Netherlands.

https://www.youtube.com/watch?v=W2_TpDcess8

Instrumentation

prometheus/client_golang

```go
httpRequestDurations = prometheus.NewHistogram(prometheus.HistogramOpts{
    Name:    "http_durations_seconds",
    Help:    "HTTP latency distribution.",
    Buckets: []float64{.005, .01, .025, .05, .1, .25, .5, 1, 2.5, 5, 10},
})
```

prometheus/client_golang

```go
httpRequestDurations = prometheus.NewHistogram(prometheus.HistogramOpts{
     Name:    "http_durations_seconds",
     Help:    "HTTP latency distribution.",
     Buckets: []float64{.005, .01, .025, .05, .1, .25, .5, 1, 2.5, 5, 10},
     NativeHistogramBucketFactor: 1.1,
})
```

prometheus/client_golang

```
httpRequestDurations = prometheus.NewHistogram(prometheus.HistogramOpts{
    Name:                         "http_durations_seconds",
    Help:                         "HTTP latency distribution.",
    NativeHistogramBucketFactor: 1.1,
})
```

prometheus/client_golang

```
httpRequestDurations = prometheus.NewHistogram(prometheus.HistogramOpts{
    Name:                            "http_durations_seconds",
    Help:                            "HTTP latency distribution.",
    NativeHistogramBucketFactor:     1.1,
    NativeHistogramMaxBucketNumber:  160,
    NativeHistogramMinResetDuration: time.Hour,
})
```

prometheus/client_golang

```go
httpRequestDurations = prometheus.NewHistogramVec(
	prometheus.HistogramOpts{
		Name:                            "http_durations_seconds",
		Help:                            "HTTP latency distribution.",
		NativeHistogramBucketFactor:     1.1,
		NativeHistogramMaxBucketNumber:  160,
		NativeHistogramMinResetDuration: time.Hour,
	},
	[]string{"status"},
)
```

prometheus/client_java

```
static final Histogram httpRequestDurations = Histogram.build()
      .name("http_durations_seconds")
      .help("HTTP latency distribution.")
      .nativeOnly()
      .nativeInitialSchema(3)
      .nativeMaxNumberOfBuckets(160)
      .nativeResetDuration(1, TimeUnit.HOURS)
      .register();
```

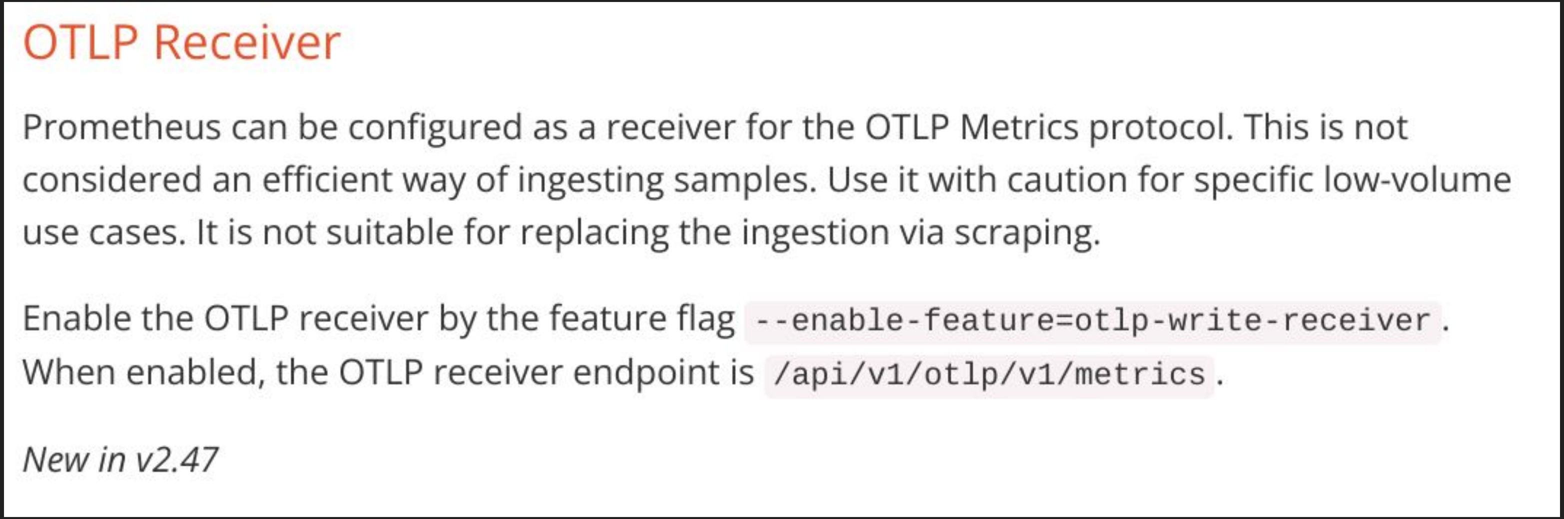
prometheus/client_*

😢

OpenTelemetry

□

Open Telemetry



OTLP Receiver

Prometheus can be configured as a receiver for the OTLP Metrics protocol. This is not considered an efficient way of ingesting samples. Use it with caution for specific low-volume use cases. It is not suitable for replacing the ingestion via scraping.

Enable the OTLP receiver by the feature flag `--enable-feature=otlp-write-receiver`. When enabled, the OTLP receiver endpoint is `/api/v1/otlp/v1/metrics`.

New in v2.47

Prometheus server configuration

```
$ prometheus --enable-feature=native-histograms
```

Introduced in v2.40.

Most recent version strongly recommended.

Optional changes of the configuration file:

Send native histograms via remote write:

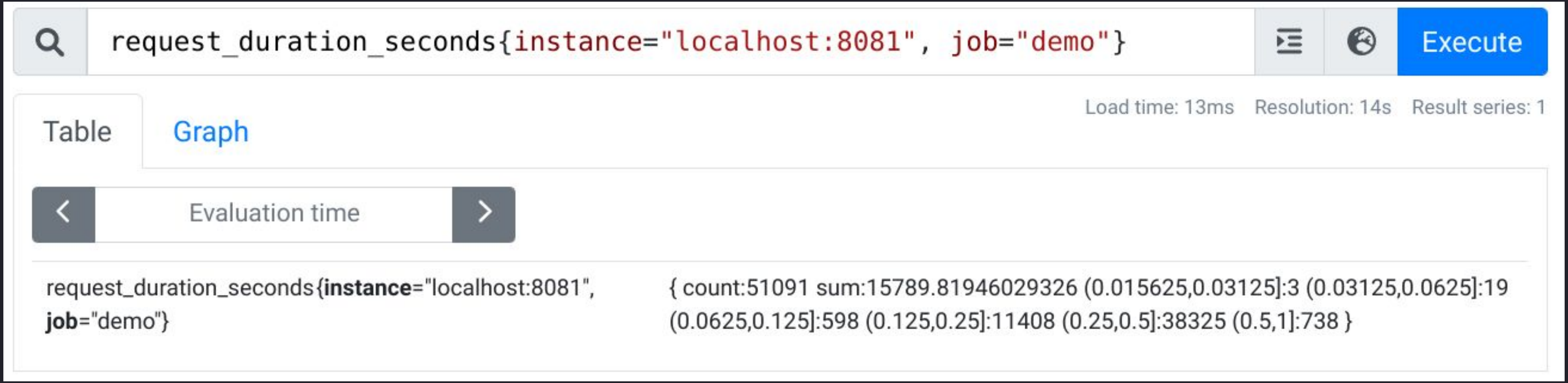
```
remote_write:
  - url: http://.../api/prom/push
    send_native_histograms: true
```

Scrape both classic and native histograms as a migration strategy:

```
scrape_configs:
  - job_name: myapp
    scrape_classic_histograms: true
```

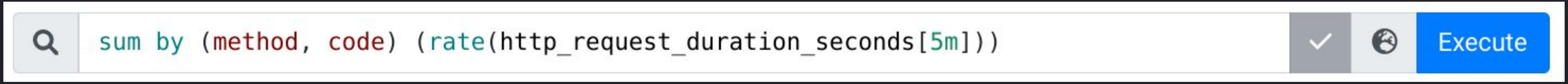
PromQL

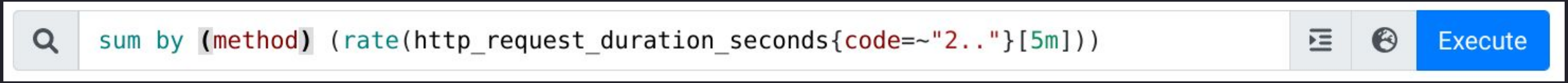
Instant Vector

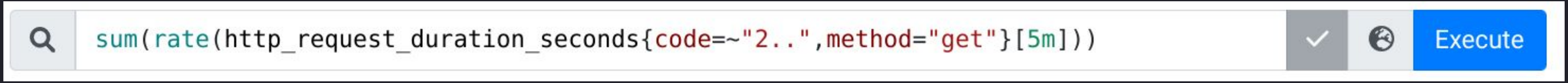

```
request_duration_seconds{instance="localhost:8081", job="demo"}
```

Load time: 13ms Resolution: 14s Result series: 1

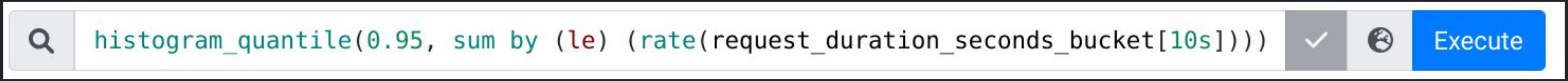
```
request_duration_seconds{instance="localhost:8081", job="demo"}    { count:51091 sum:15789.81946029326 (0.015625,0.03125]:3 (0.03125,0.0625]:19 (0.0625,0.125]:598 (0.125,0.25]:11408 (0.25,0.5]:38325 (0.5,1]:738 }
```


```
sum by (method, code) (rate(http_request_duration_seconds[5m]))
```

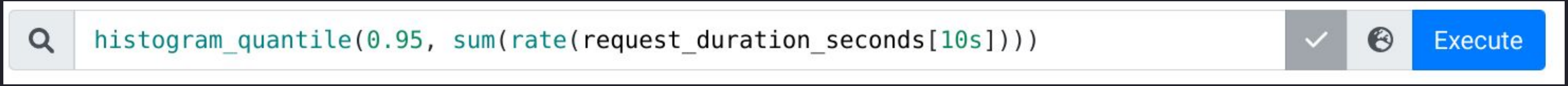

```
sum by (method) (rate(http_request_duration_seconds{code=~"2.."}[5m]))
```


```
sum(rate(http_request_duration_seconds{code=~"2..",method="get"}[5m]))
```

Quantile estimation

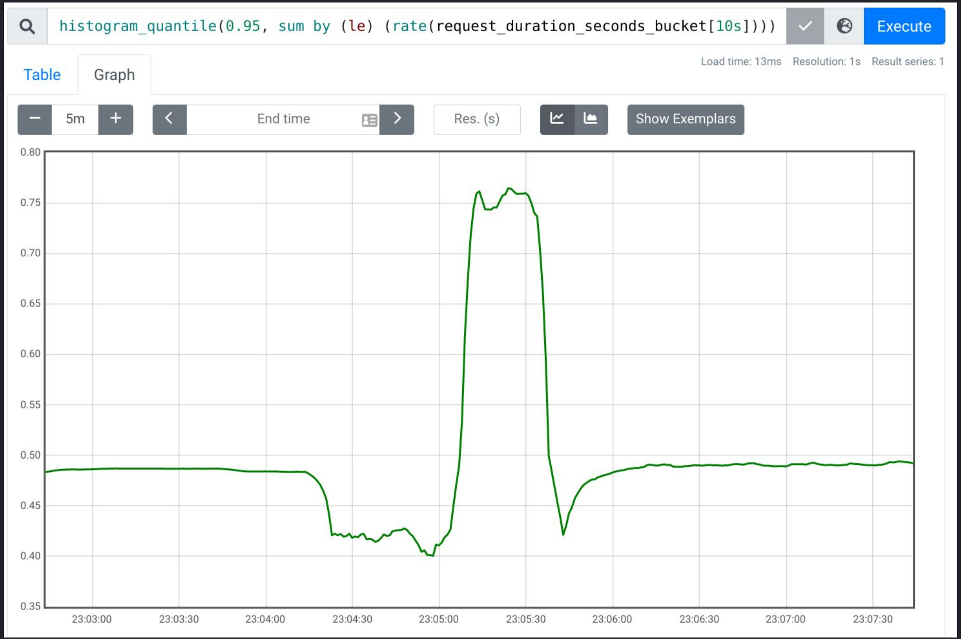


```
histogram_quantile(0.95, sum by (le) (rate(request_duration_seconds_bucket[10s])))
```



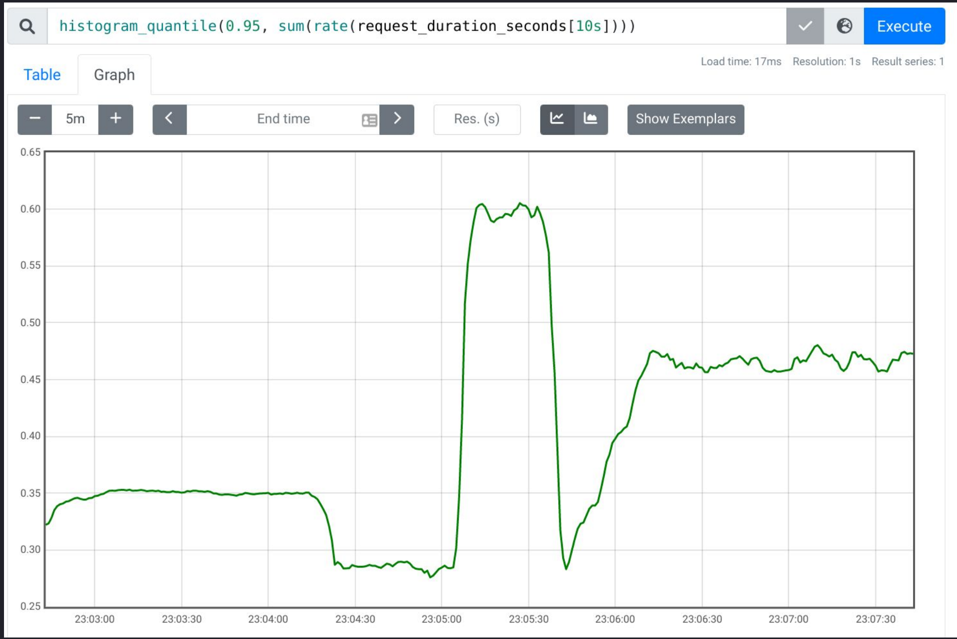
```
histogram_quantile(0.95, sum(rate(request_duration_seconds[10s])))
```

Quantile estimation



```
histogram_quantile(0.95, sum by (le) (rate(request_duration_seconds_bucket[10s])))
```

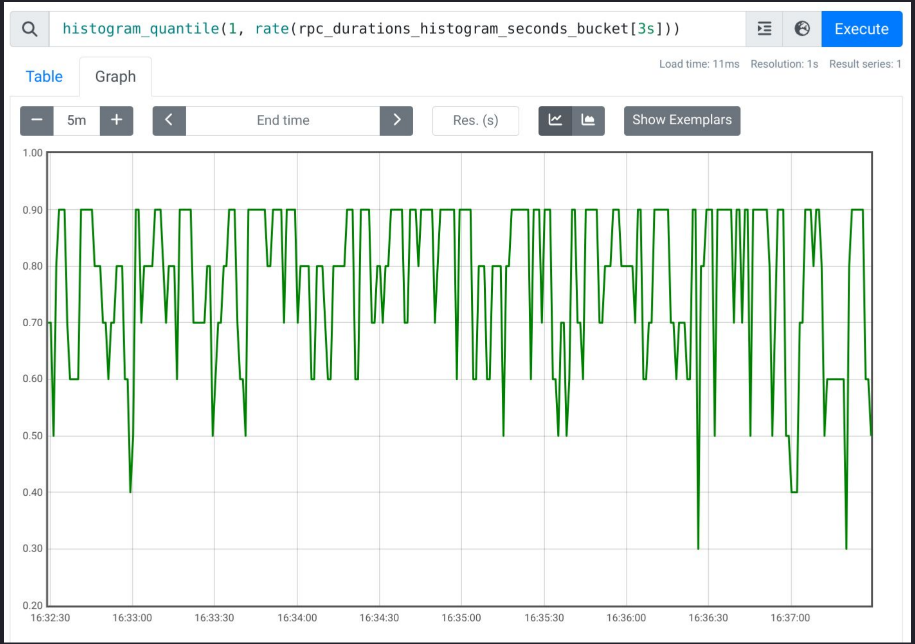
Load time: 13ms Resolution: 1s Result series: 1



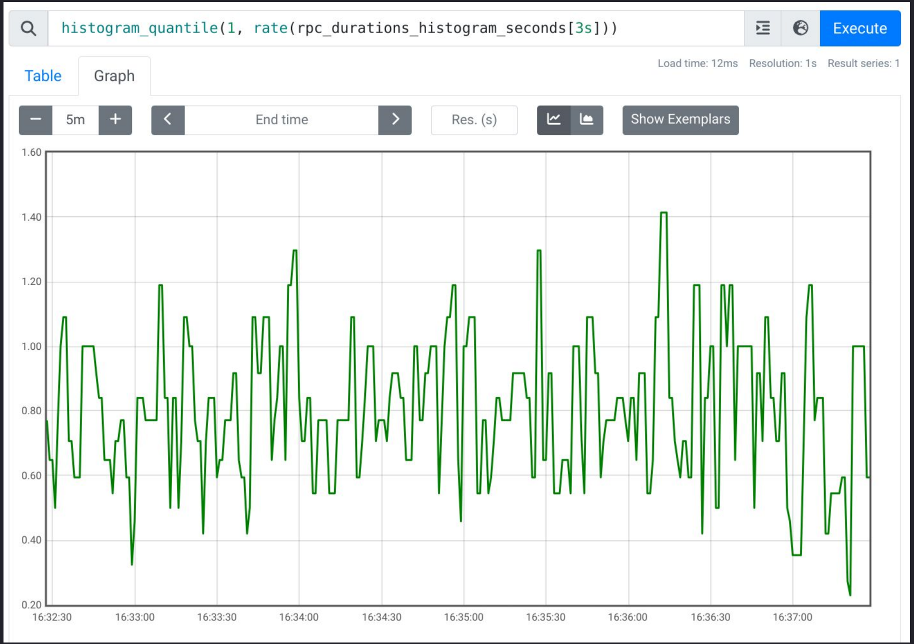
```
histogram_quantile(0.95, sum(rate(request_duration_seconds[10s])))
```

Load time: 17ms Resolution: 1s Result series: 1

Maximum AKA 100th percentile

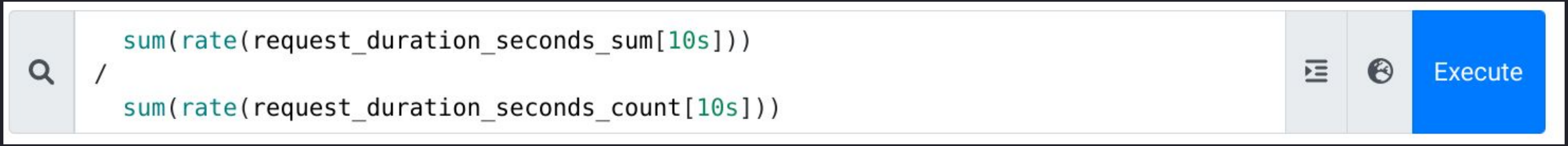


```
histogram_quantile(1, rate(rpc_durations_histogram_seconds_bucket[3s]))
```

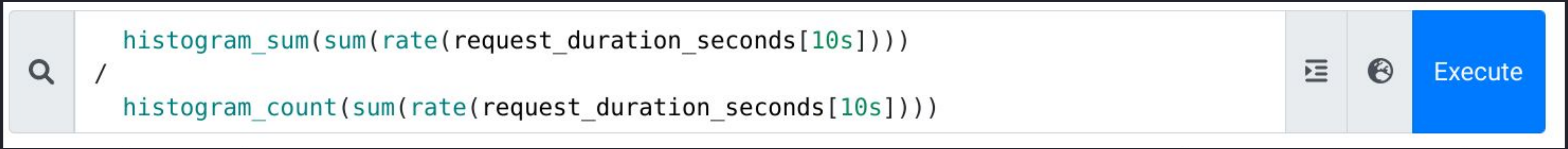


```
histogram_quantile(1, rate(rpc_durations_histogram_seconds[3s]))
```

Average calculation

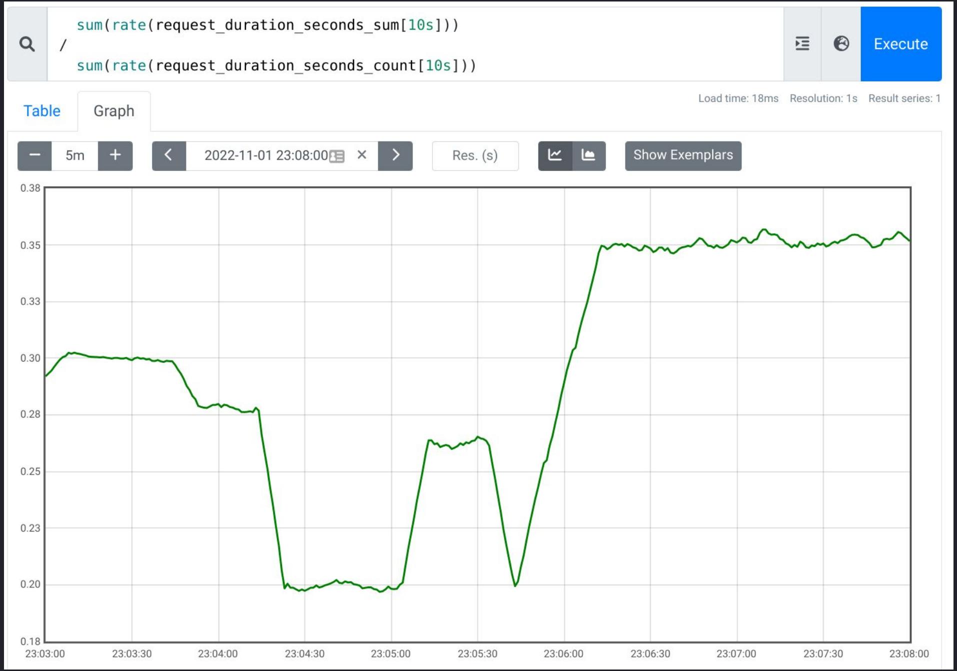


```
  sum(rate(request_duration_seconds_sum[10s]))
/
  sum(rate(request_duration_seconds_count[10s]))
```



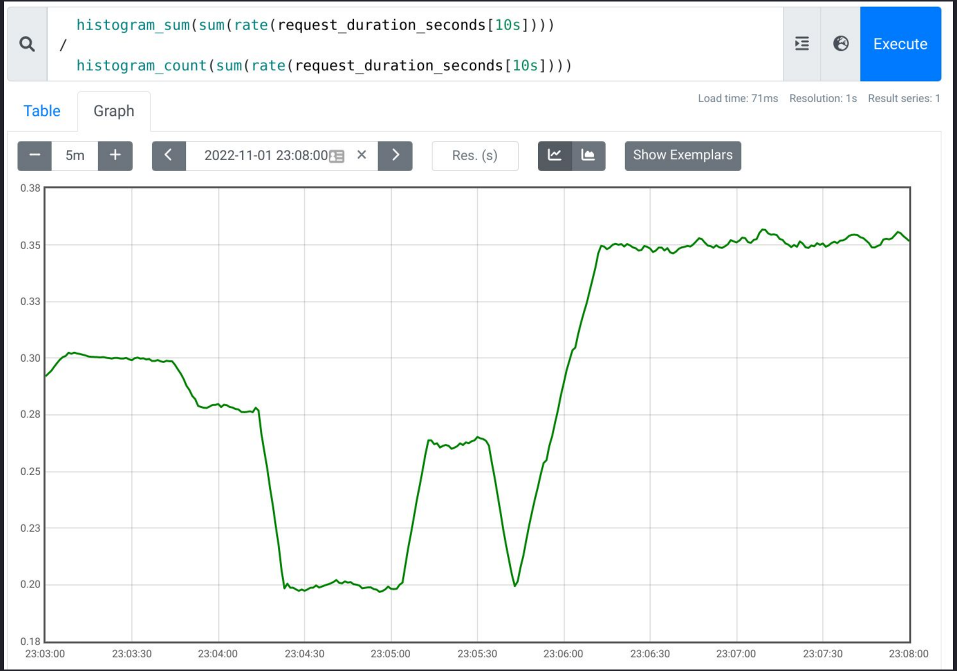
```
  histogram_sum(sum(rate(request_duration_seconds[10s])))
/
  histogram_count(sum(rate(request_duration_seconds[10s])))
```

Average calculation



```
  sum(rate(request_duration_seconds_sum[10s]))
/
  sum(rate(request_duration_seconds_count[10s]))
```

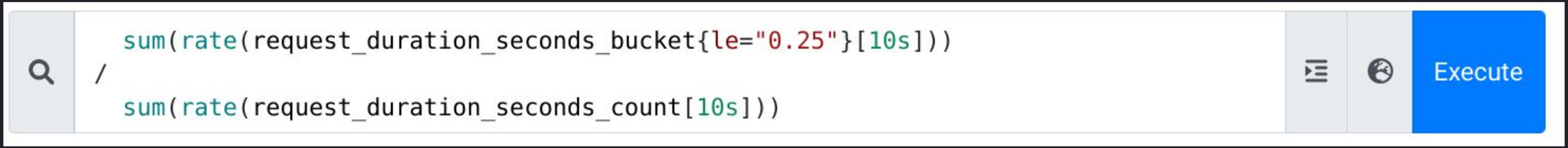
Load time: 18ms Resolution: 1s Result series: 1



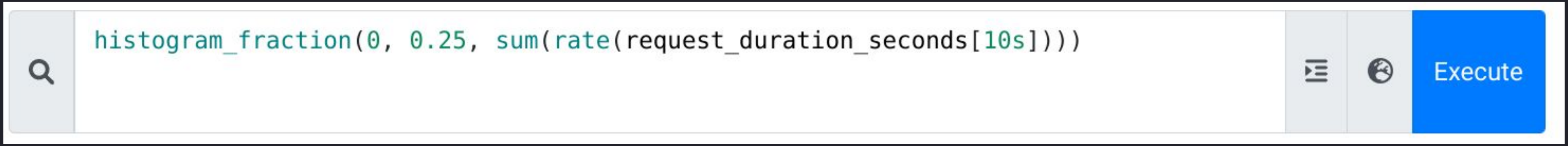
```
  histogram_sum(sum(rate(request_duration_seconds[10s])))
/
  histogram_count(sum(rate(request_duration_seconds[10s])))
```

Load time: 71ms Resolution: 1s Result series: 1

Fraction ~~calculation~~ estimation

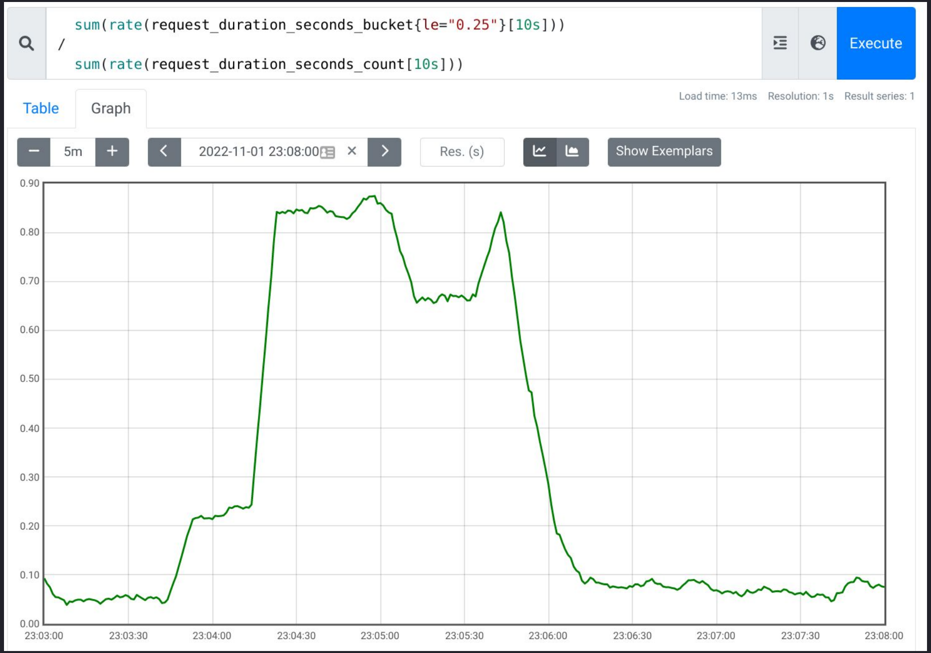


```
  sum(rate(request_duration_seconds_bucket{le="0.25"}[10s]))
/
  sum(rate(request_duration_seconds_count[10s]))
```

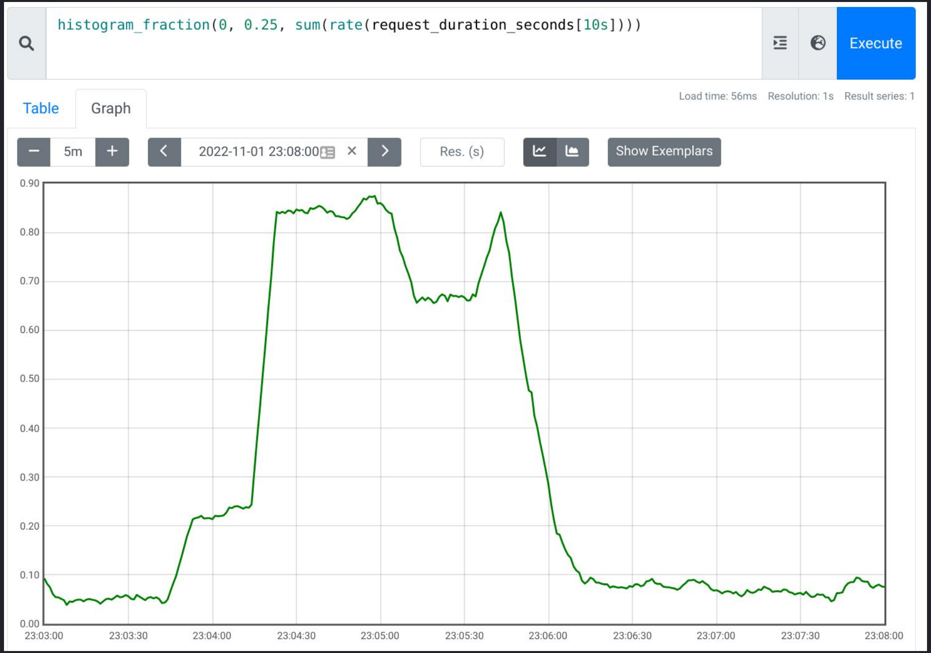


```
histogram_fraction(0, 0.25, sum(rate(request_duration_seconds[10s])))
```

Fraction ~~calculation~~ estimation

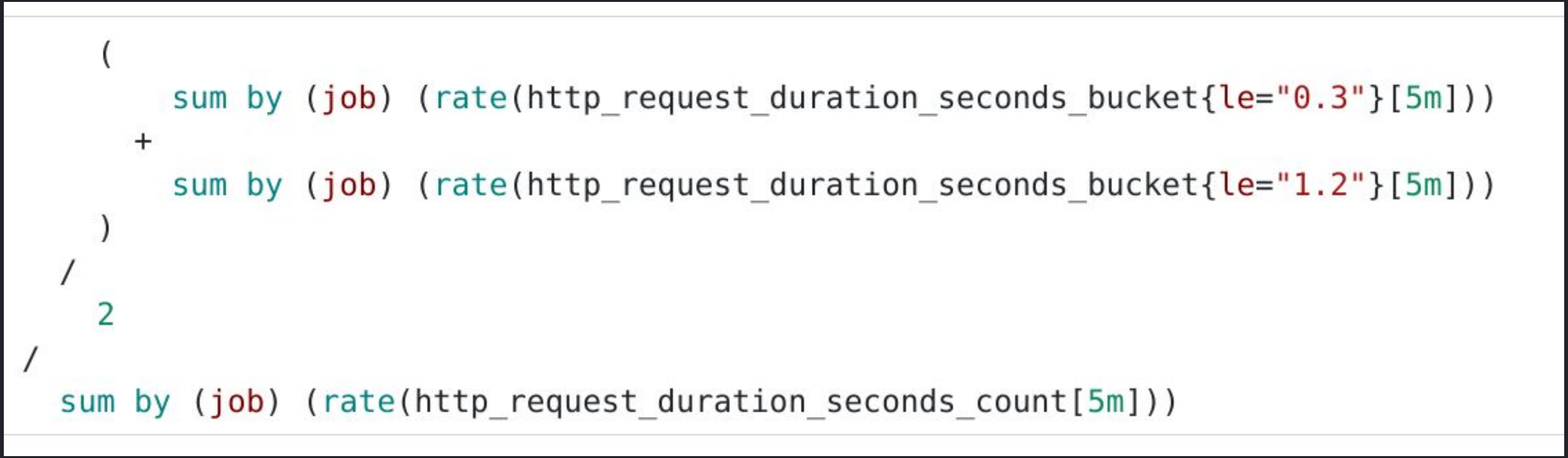


```
  sum(rate(request_duration_seconds_bucket{le="0.25"}[10s]))
/
  sum(rate(request_duration_seconds_count[10s]))
```

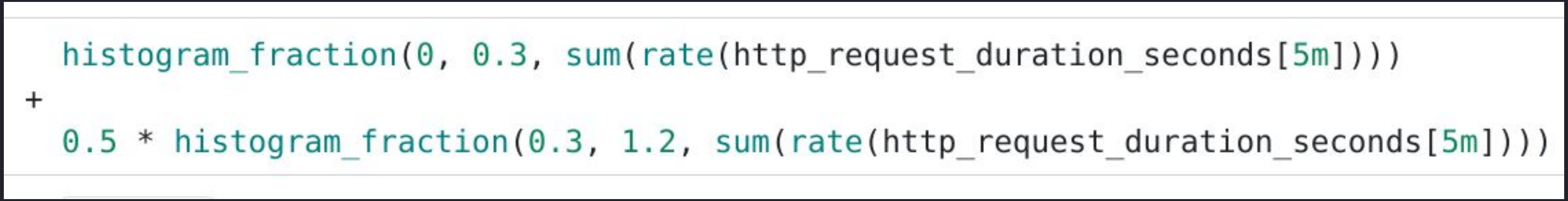


```
histogram_fraction(0, 0.25, sum(rate(request_duration_seconds[10s])))
```

Apdex score https://en.wikipedia.org/wiki/Apdex



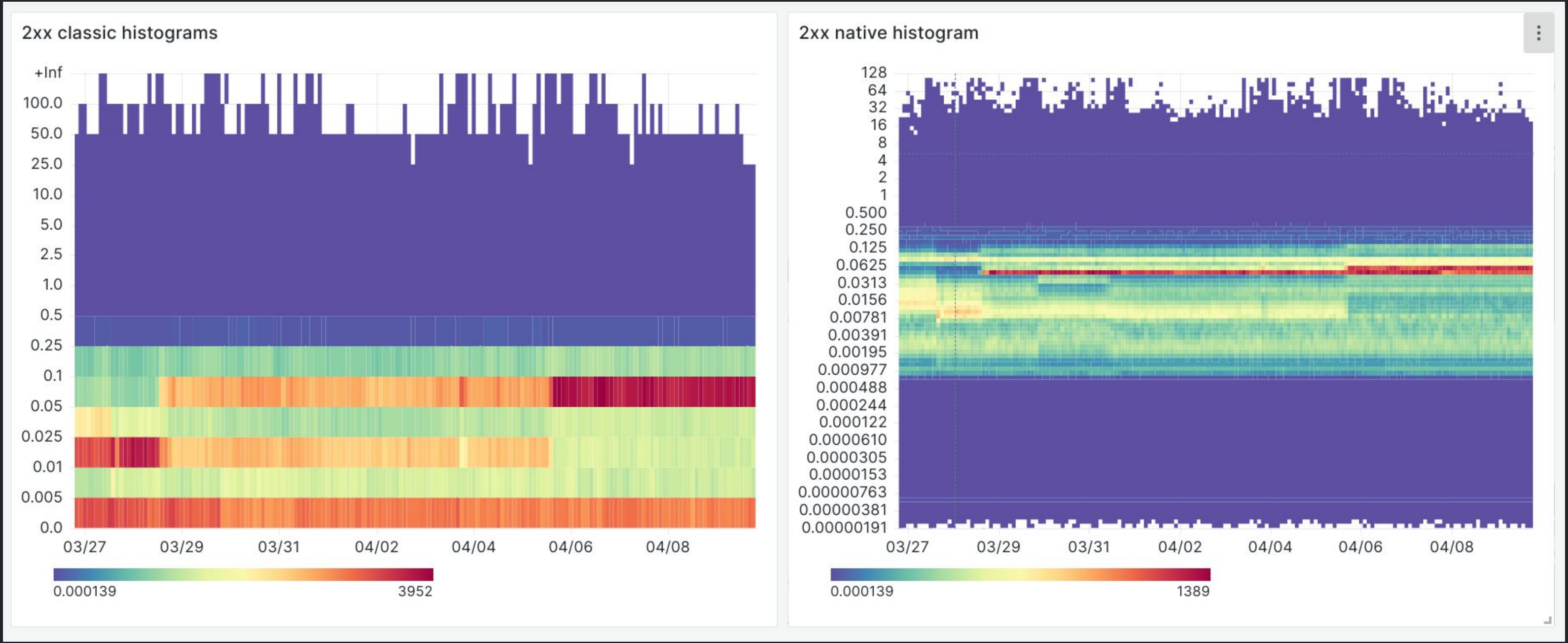
```
    (
        sum by (job) (rate(http_request_duration_seconds_bucket{le="0.3"}[5m]))
      +
        sum by (job) (rate(http_request_duration_seconds_bucket{le="1.2"}[5m]))
    )
  /
    2
/
  sum by (job) (rate(http_request_duration_seconds_count[5m]))
```

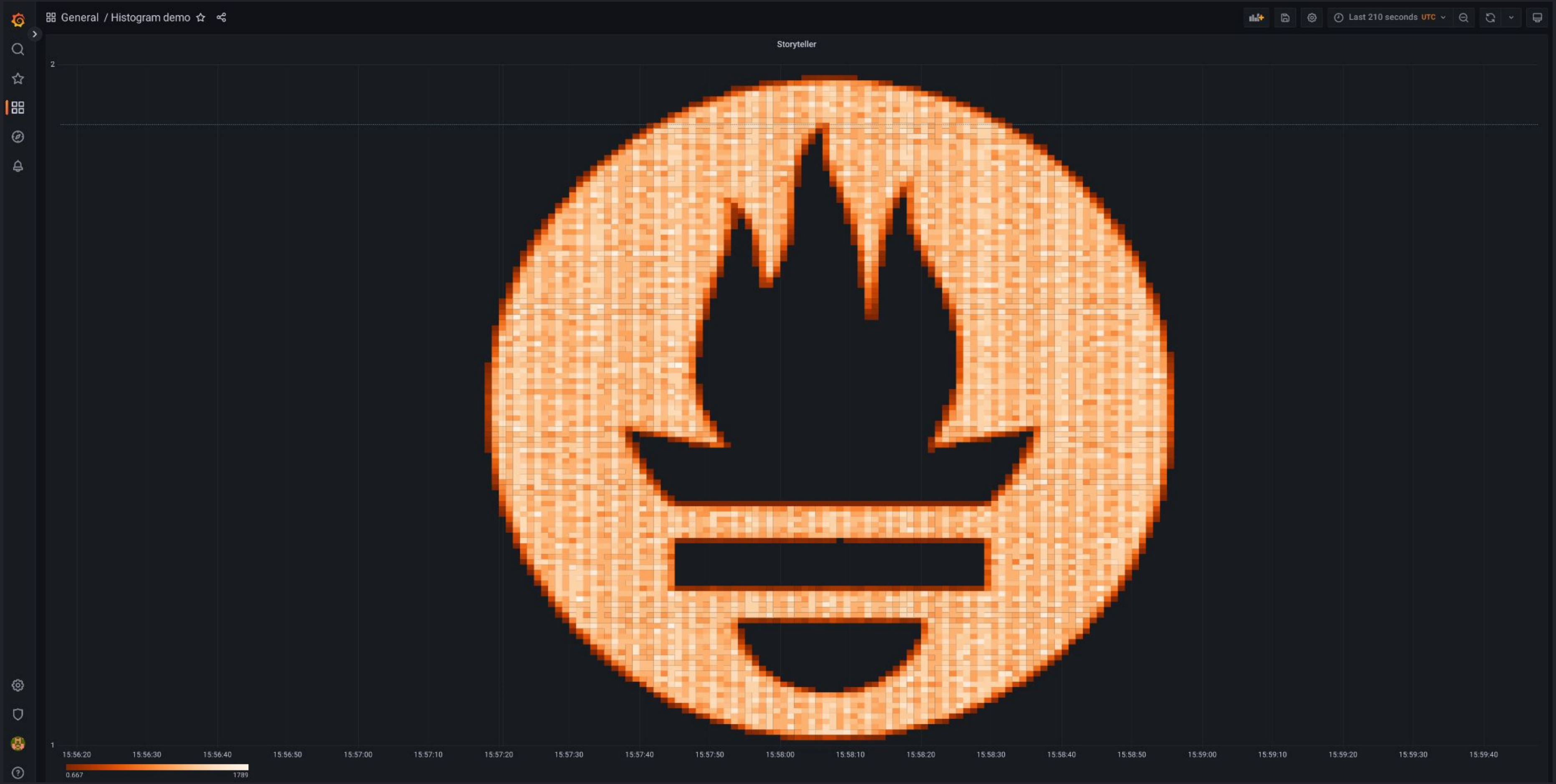


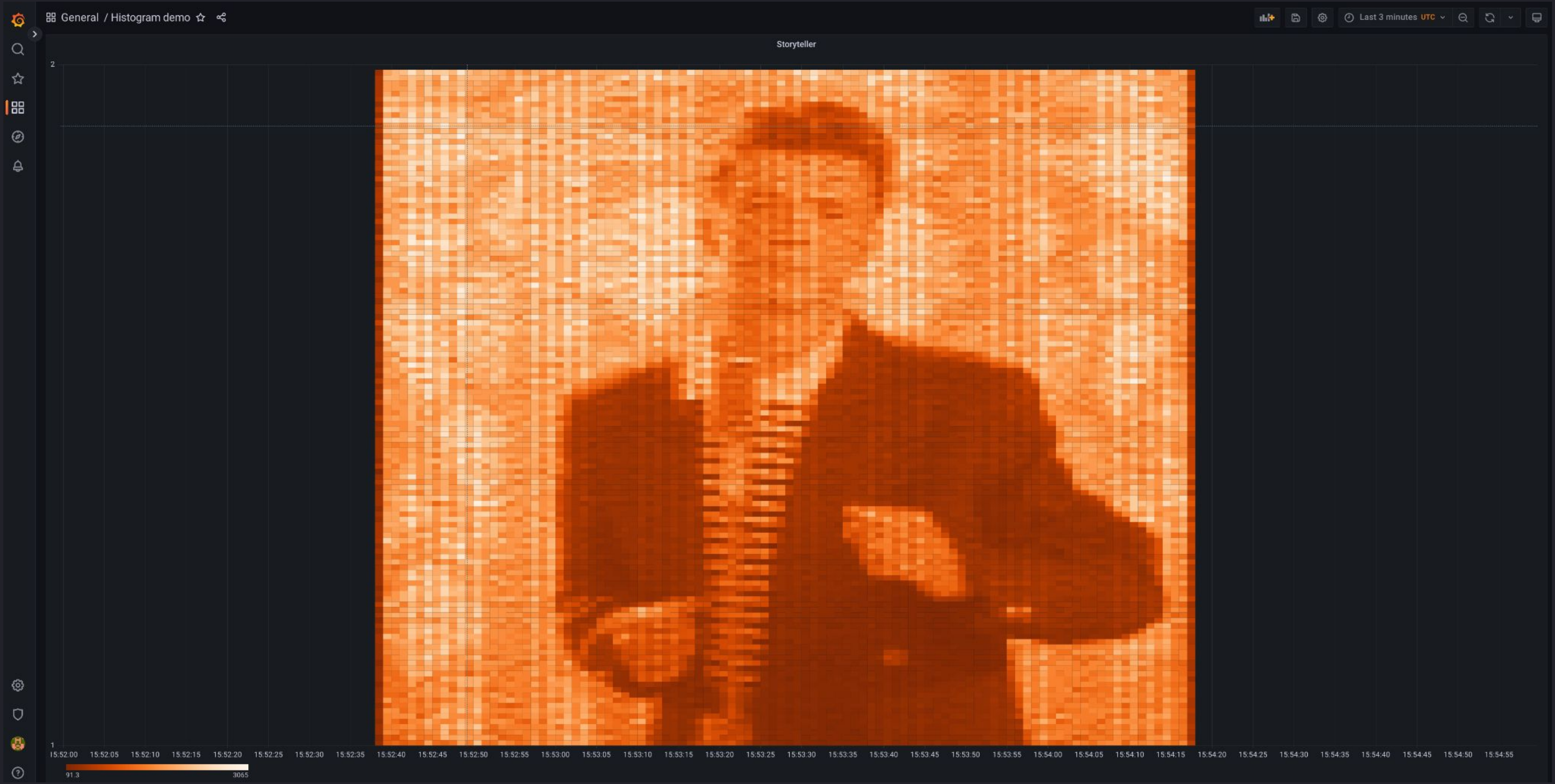
```
  histogram_fraction(0, 0.3, sum(rate(http_request_duration_seconds[5m])))
+
  0.5 * histogram_fraction(0.3, 1.2, sum(rate(http_request_duration_seconds[5m])))
```

Heatmaps

Heatmaps


2xx classic histograms
+Inf
100.0
50.0
25.0
10.0
5.0
2.5
1.0
0.5
0.25
0.1
0.05
0.025
0.01
0.005
0.0
03/27
03/29
03/31
04/02
04/04
04/06
04/08
0.000139
3952
2xx native histogram
128
64
32
16
8
4
2
1
0.500
0.250
0.125
0.0625
0.0313
0.0156
0.00781
0.00391
0.00195
0.000977
0.000488
0.000244
0.000122
0.0000610
0.0000305
0.0000153
0.00000763
0.00000381
0.00000191
03/27
03/29
03/31
04/02
04/04
04/06
04/08
0.000139
1389


General / Histogram demo
Last 210 seconds UTC
Storyteller
2
1
15:56:20
15:56:30
15:56:40
15:56:50
15:57:00
15:57:10
15:57:20
15:57:30
15:57:40
15:57:50
15:58:00
15:58:10
15:58:20
15:58:30
15:58:40
15:58:50
15:59:00
15:59:10
15:59:20
15:59:30
15:59:40
0.667
1789


General / Histogram demo
Last 3 minutes UTC
Storyteller
2
1
15:52:00
15:52:05
15:52:10
15:52:15
15:52:20
15:52:25
15:52:30
15:52:35
15:52:40
15:52:45
15:52:50
15:52:55
15:53:00
15:53:05
15:53:10
15:53:15
15:53:20
15:53:25
15:53:30
15:53:35
15:53:40
15:53:45
15:53:50
15:53:55
15:54:00
15:54:05
15:54:10
15:54:15
15:54:20
15:54:25
15:54:30
15:54:35
15:54:40
15:54:45
15:54:50
15:54:55
91.3
3065

Downsides

- Expect bugs.
- Things might still change.
- Some PromQL evaluations are still slower.
- Exponential bucketing might be a misfit.
- Cannot pick arbitrary bucket boundaries.

Future

- Better graphical representation in UI.
- Instrumentation in more languages.
- Custom bucket layouts.

https://github.com/prometheus/prometheus/milestone/10

Grafana Labs

https://github.com/beorn7/talks

beorn@grafana.com