SLOs With Prometheus: Done Wrong, Wrong, Wrong, *Wrong*, Then Right

Carson Anderson
DevX-O, Weave

@carsonoid

@carsonoid@kind.social

weave

getweave.com

Good News,
Other Good News

https://engineering.getweave.com/

https://engineering.getweave.com/post/slos-wrong-wrong-wrong-right/

* SLO Basics Presentation *

https://engineering.getweave.com/talk/slos-in-practice-and-at-scale/

Where did it start?

Data Overload!

SLOs To The Rescue!

What is an SLO?

Google: https://sre.google/sre-book/service-level-objectives/

An SLO is a **service level objective**: a target value or range of values for a service level that is measured by an SLI

Carson

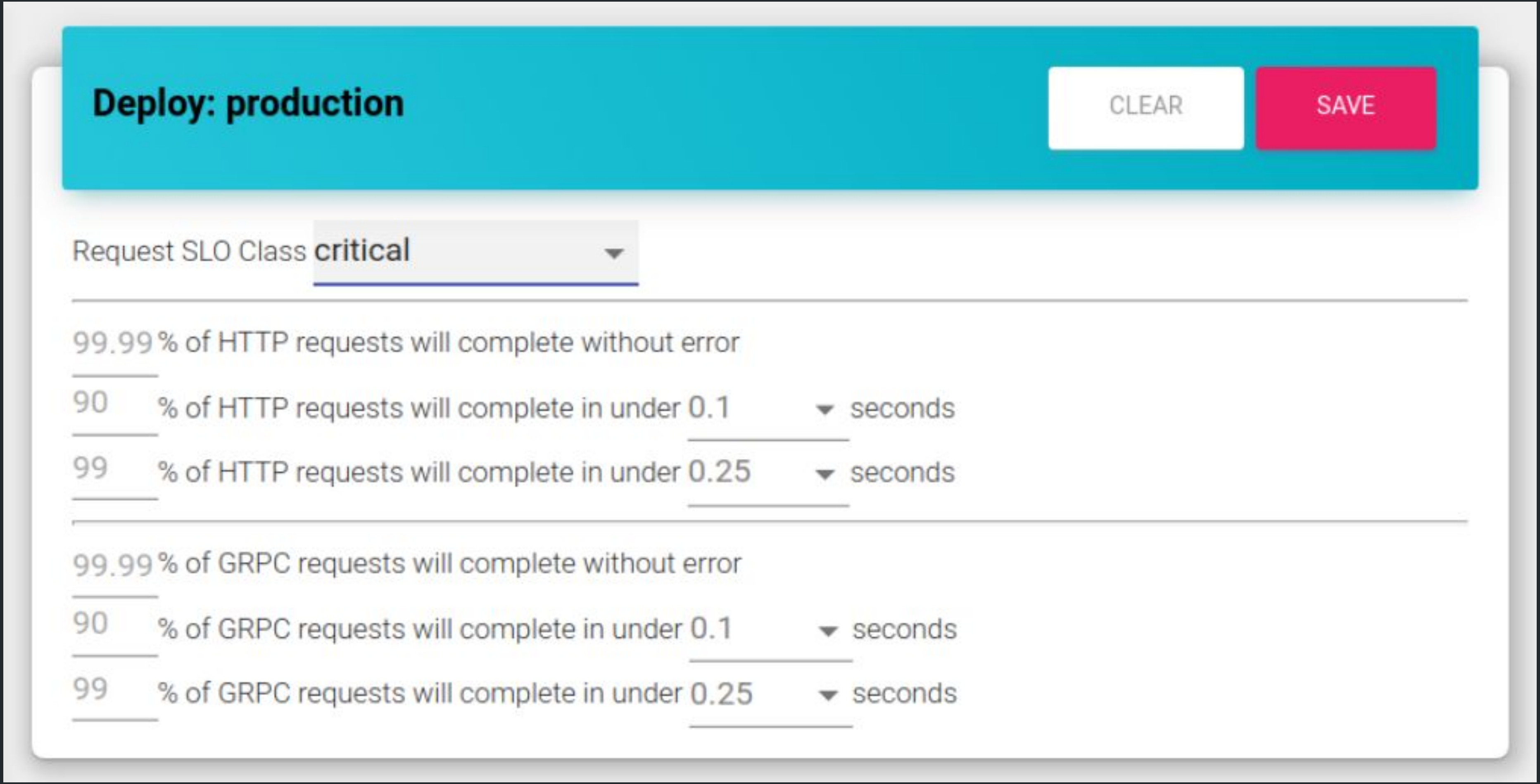
An SLO is a **service level objective**: a measure of "normal" performance for a system https://sre.google/sre-book/service-level-objectives/

https://sre.google/workbook/implementing-slos/

Objective

"99.9% of HTTP requests will complete without a 500 level error"

What Was The Vision?


Deploy: production

CLEAR SAVE

Request SLO Class critical

99.99 % of HTTP requests will complete without error

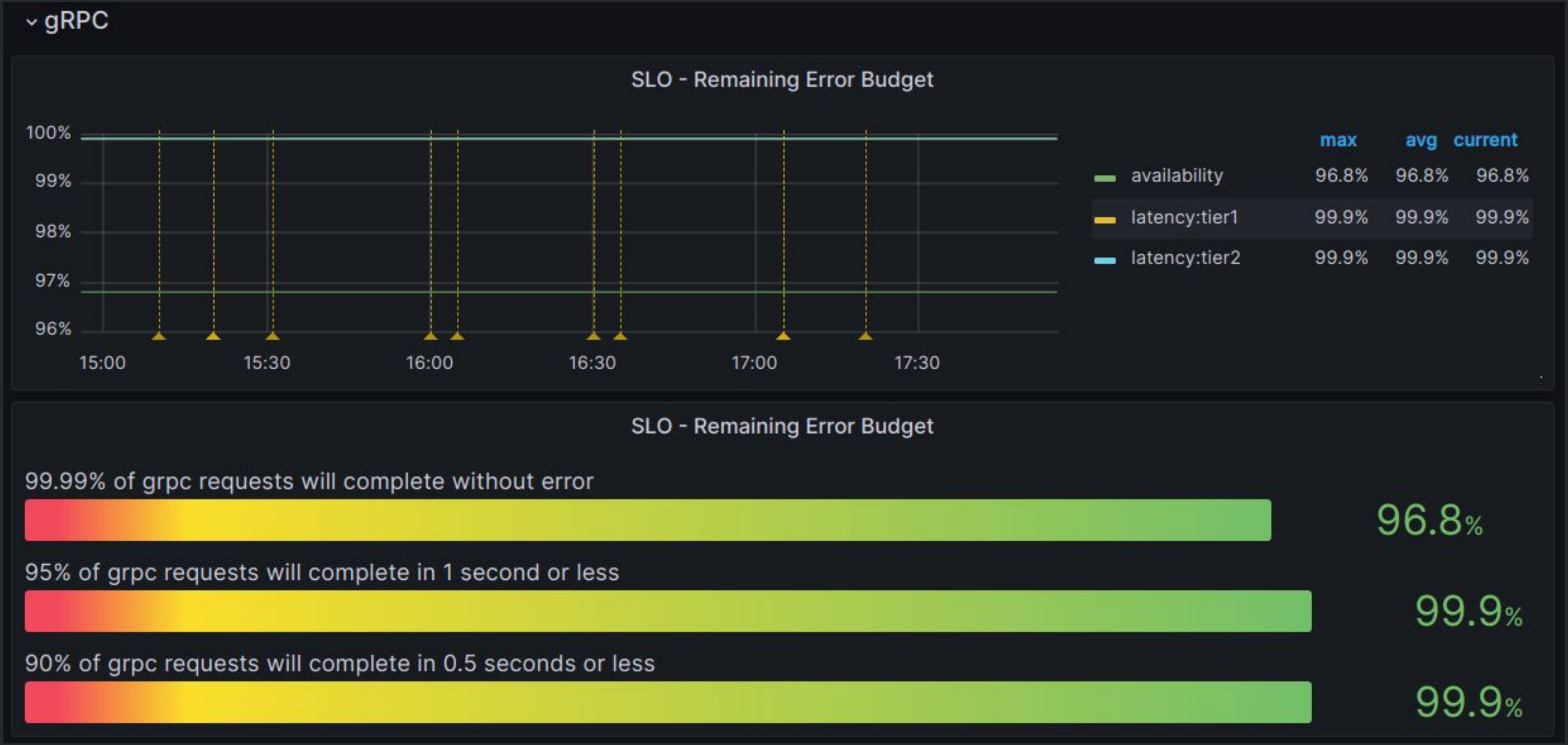
90 % of HTTP requests will complete in under 0.1 seconds

99 % of HTTP requests will complete in under 0.25 seconds

99.99 % of GRPC requests will complete without error

90 % of GRPC requests will complete in under 0.1 seconds

99 % of GRPC requests will complete in under 0.25 seconds



	max	avg	current
🟩 availability	96.8%	96.8%	96.8%
🟨 latency:tier1	99.9%	99.9%	99.9%
🟦 latency:tier2	99.9%	99.9%	99.9%

SLO - Remaining Error Budget

99.99% of grpc requests will complete without error — 96.8%

95% of grpc requests will complete in 1 second or less — 99.9%

90% of grpc requests will complete in 0.5 seconds or less — 99.9%

SLO Remaining Budget Formula

$$\frac{\text{Budgeted} - \text{Actual}}{\text{Budgeted}} = \text{\% Budget Remaining}$$

Example Data Points

- 10,000 total HTTP Requests handled in the last 28 days
- 20 of the requests have had a server error in the last 28 days

Example Calculations

- 99.9% in decimal = .01
- 10,000 * .01 = 100 Budgeted Failures

$$\frac{\text{Budgeted} - \text{Actual}}{\text{Budgeted}} = \text{\% Budget Remaining}$$

$$\frac{100 - 20}{100} = \begin{matrix} 0.8 \\ 80\% \end{matrix}$$

Existing Prometheus Metrics

- http_timer_bucket - A histogram of request times
- http_timer_count - A counter for requests
- http_timer_sum   - A counter for total time processing

- grpc_timer_bucket - A histogram of request times
- grpc_timer_count - A counter for requests
- grpc_timer_sum   - A counter for total time processing

Existing Scale

- All services scraped every minute
- About 200 services serving gRPC
  - Over 400,000 data points per scrape
- About 160 services serving HTTP
  - Over 200,000 data points per scrape

864 Million Data Points per day!

Attempt 1

A Big 'Ol Query!

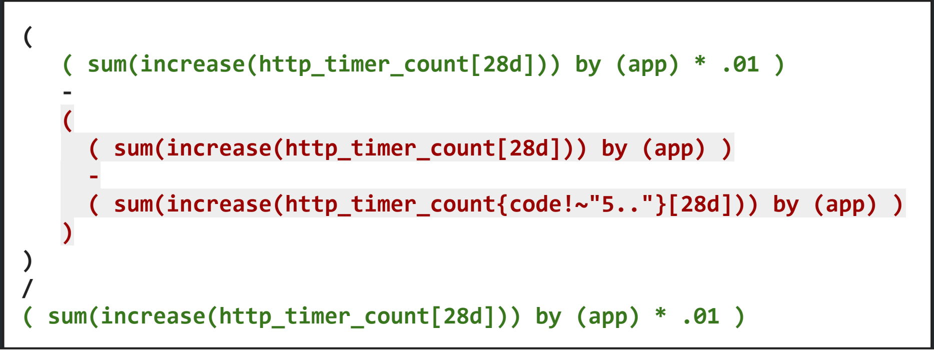
SLOs for Everyone, All At Once

$$\frac{\text{Budgeted} - \text{Actual}}{\text{Budgeted}} = \text{\% Budget Remaining}$$

.01 = 1-.99 = 99%

```
(
   ( sum(increase(http_timer_count[28d])) by (app) * .01 )
   -
   (
     ( sum(increase(http_timer_count[28d])) by (app) )
     -
     ( sum(increase(http_timer_count{code!~"5.."}[28d])) by (app) )
   )
)
/
( sum(increase(http_timer_count[28d])) by (app) * .01 )
```

Total - Passed = Failed 👎 Don't do this.
Just use "or vector(0)" for sparse metrics



```
(
   ( sum(increase(http_timer_count[28d])) by (app) * .01 )
   -
   (
     ( sum(increase(http_timer_count[28d])) by (app) )
     -
     ( sum(increase(http_timer_count{code!~"5.."}[28d])) by (app) )
   )
)
/
( sum(increase(http_timer_count[28d])) by (app) * .01 )
```

Good 👍

- One Query To Write

Bad 👎

- Query never completed
- All apps have to have the same Objective (99%)

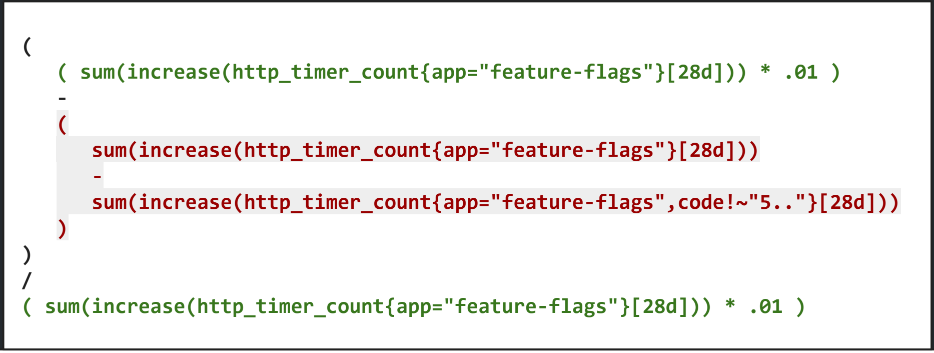
Attempt 2

A Smaller Big 'Ol Query!

(One Service At A Time)

$$\frac{\text{Budgeted} - \text{Actual}}{\text{Budgeted}} = \text{\% Budget Remaining}$$

```
(
   ( sum(increase(http_timer_count{app="feature-flags"}[28d])) * .01 )
   -
   (
     sum(increase(http_timer_count{app="feature-flags"}[28d]))
     -
     sum(increase(http_timer_count{app="feature-flags",code!~"5.."}[28d]))
   )
)
/
( sum(increase(http_timer_count{app="feature-flags"}[28d])) * .01 )
```



```
(
   ( sum(increase(http_timer_count{app="feature-flags"}[28d])) * .01 )
   -
   (
      sum(increase(http_timer_count{app="feature-flags"}[28d]))
      -
      sum(increase(http_timer_count{app="feature-flags",code!~"5.."}[28d]))
   )
)
/
( sum(increase(http_timer_count{app="feature-flags"}[28d])) * .01 )
```

Good 👍

- Query Completes!

Bad 👎

- Query **Eventually** Completes
  - About 8.5 seconds for one result
- Query unusable in dashboards

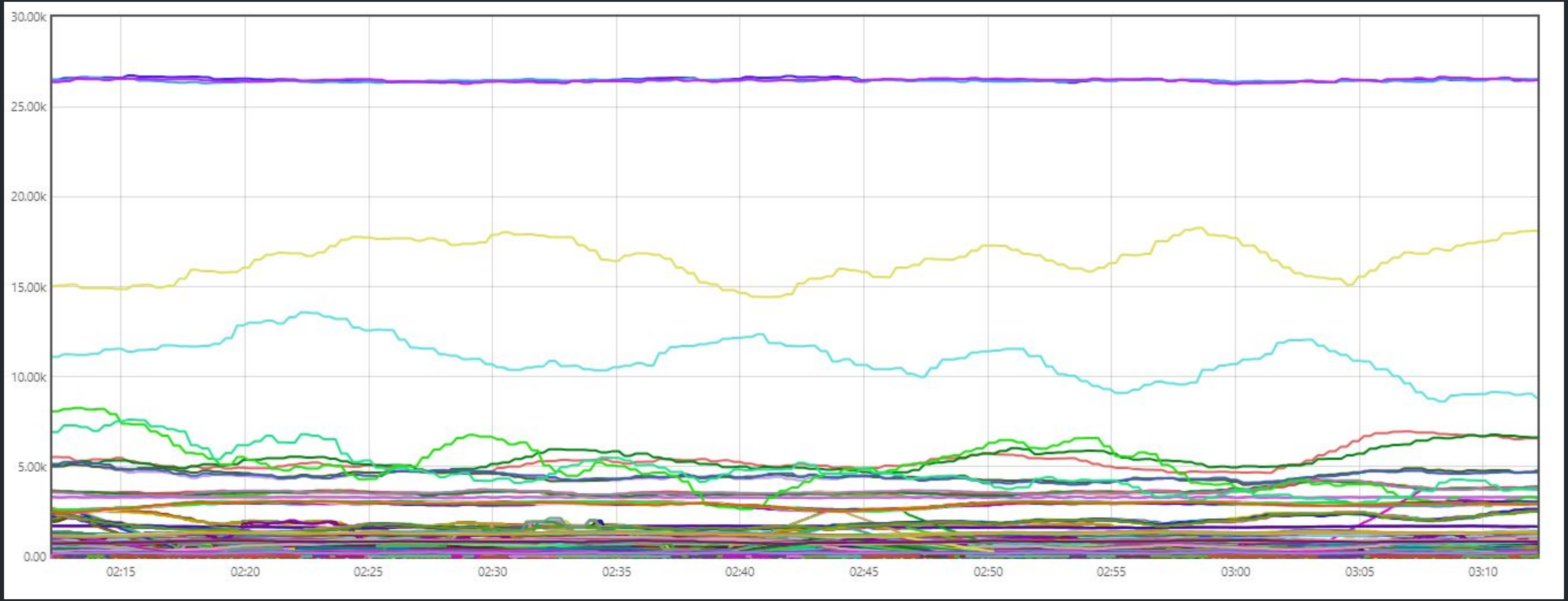
Attempt 3: Preamble

Introducing Recording Rules!

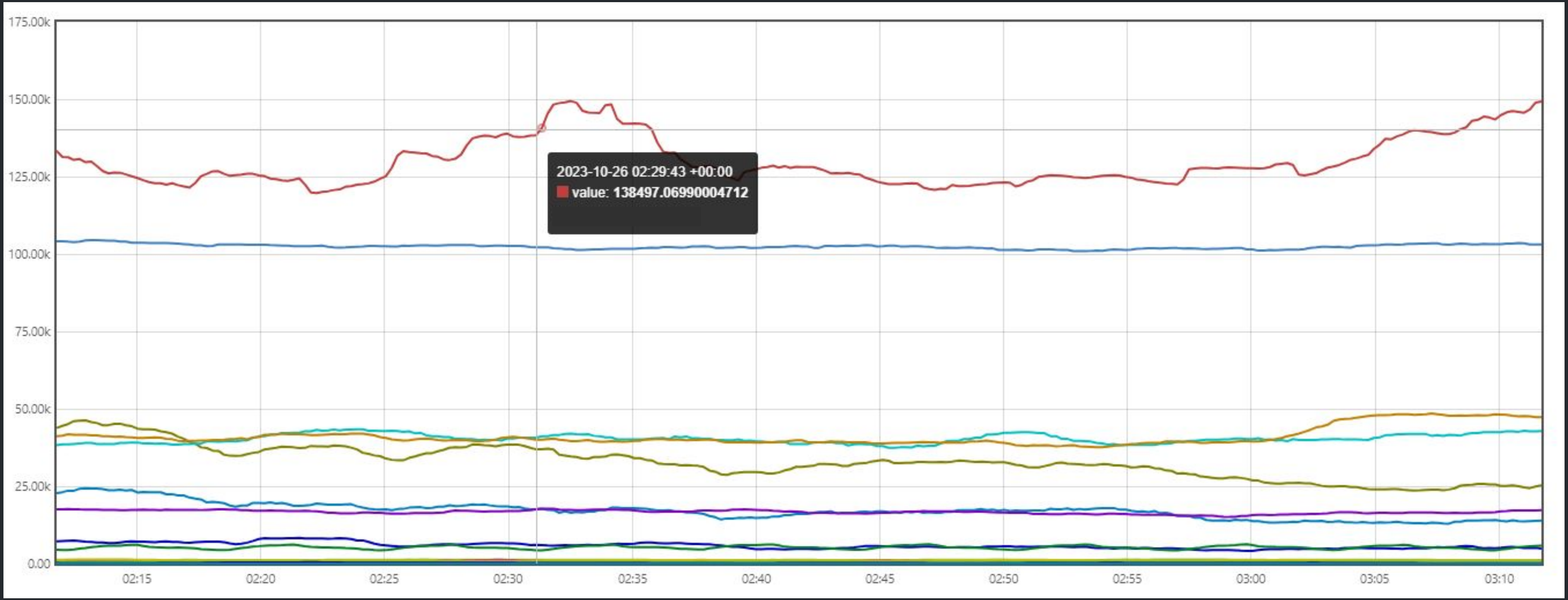
What is a recording rule?

A Prometheus query that is run on a regular basis and then saved back to the datastore as a unique data set

my_metric{}


30.00k
25.00k
20.00k
15.00k
10.00k
5.00k
0.00
02:15
02:20
02:25
02:30
02:35
02:40
02:45
02:50
02:55
03:00
03:05
03:10

sum(my_metric) by (app)

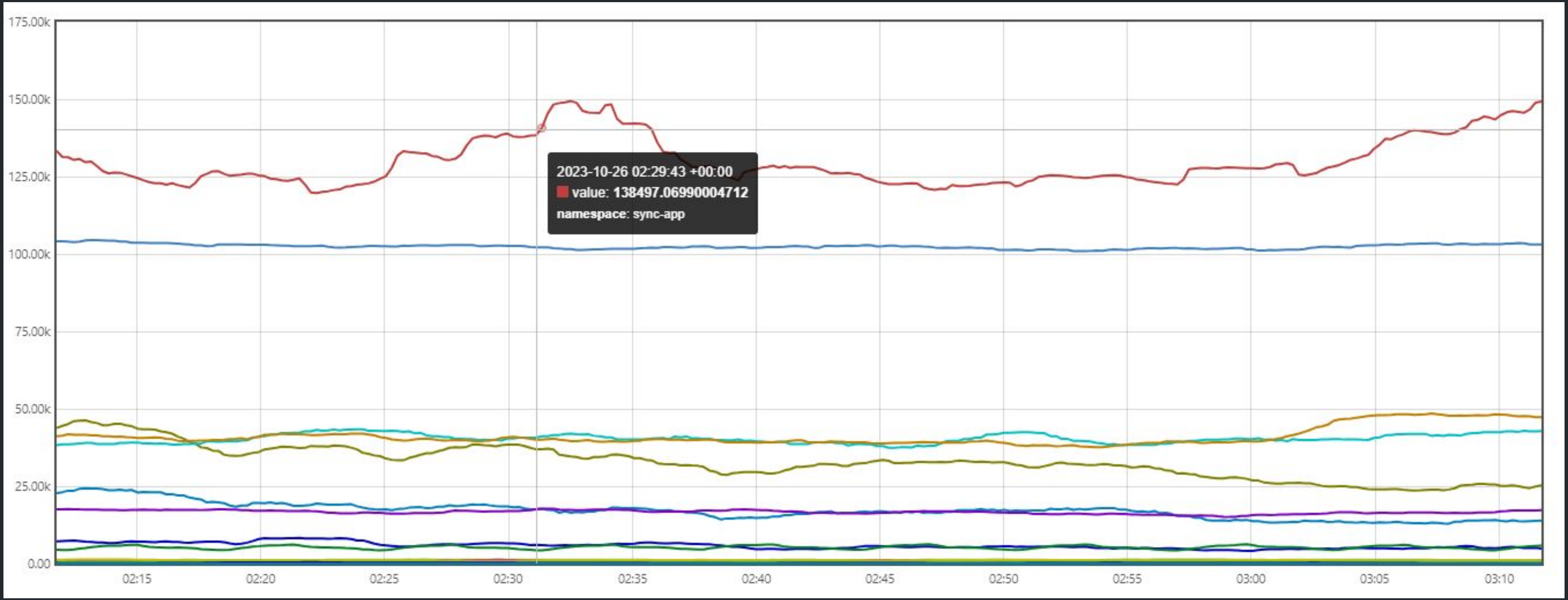


2023-10-26 02:29:43 +00:00
🟥 value: 138497.06990004712

```
groups:
- name: my-record-rules
  interval: 1m
  rules:
  - record: my_metric:sum
    query: sum(my_metric) by (app)
```

~~sum(my_metric) by (app)~~

my_metric:sum



Attempt 3

(Mis)Use Recording Rules!

Duplication == Lag

```
(
   ( sum(increase(http_timer_count{app="feature-flags"}[28d])) * .01 )
   -
   (
      sum(increase(http_timer_count{app="feature-flags"}[28d]))
      -
      sum(increase(http_timer_count{app="feature-flags",code!~"5.."}[28d]))
   )
)
/
( sum(increase(http_timer_count{app="feature-flags"}[28d])) * .01 )
```

Per-app Rule 1

```
- record: slo:feature_flags:request:http:availability:total
  expr: |
        sum(increase(http_timer_count{app="feature-flags"}[28d]))
```

Per-app Rule 2

```
- record: slo:feature_flags:request:http:availability:budgeted
  expr: |
        (slo:feature_flags:request:http:availability:total * .01)
```

Per-app Rule 3

```
- record: slo:feature_flags:request:http:availability:failed
  expr: |
    (
      slo:feature_flags:request:http:availability:total
      -
      sum(increase(http_timer_count{app="feature-flags",code!~"5.."}[28d]))
    )
```

Final Rule

$$\frac{\text{Budgeted} - \text{Actual}}{\text{Budgeted}} = \text{\% Budget Remaining}$$

```
- record: slo:feature_flags:request:http:availability:error_budget
  expr: |
    (
     slo:feature_flags:request:http:availability:budgeted -
     slo:feature_flags:request:http:availability:failed
    )
    /
    slo:feature_flags:request:http:availability:budgeted
```

All Rules For One App

```
groups:
- name: feature-flag-slo.rules
  interval: 1m
  rules:
  - record: slo:feature_flags:request:http:availability:total
    expr: |
          sum(increase(http_timer_count{app="feature-flags"}[28d]))
  - record: 'slo:feature_flags:request:http:availability:budgeted'
    expr: |
          (slo:feature_flags:request:http:availability:total * .01)
  - record: 'slo:feature_flags:request:http:availability:failed'
    expr: |
      (
        slo:feature_flags:request:http:availability:total
        -
        sum(increase(http_timer_count{app="feature-flags",code!~"5.."}[28d]))
      )
  - record: slo:feature_flags:request:http:availability:error_budget
    expr: |
      (
      slo:feature_flags:request:http:availability:budgeted -
      slo:feature_flags:request:http:availability:failed
      )
      /
      slo:feature_flags:request:http:availability:budgeted
```

Attempt 3 Results

Good 👍

- Query Completes!
- Final metric renders in dashboard

Bad 👎

- Long unique rule names break dashboard templates
- **Random inexplicable spikes in charts**
- **Rules still don't work at all for some apps**

Attempt 4

Get It Right! 🎉

Or so I thought…

System-Wide Sum metrics

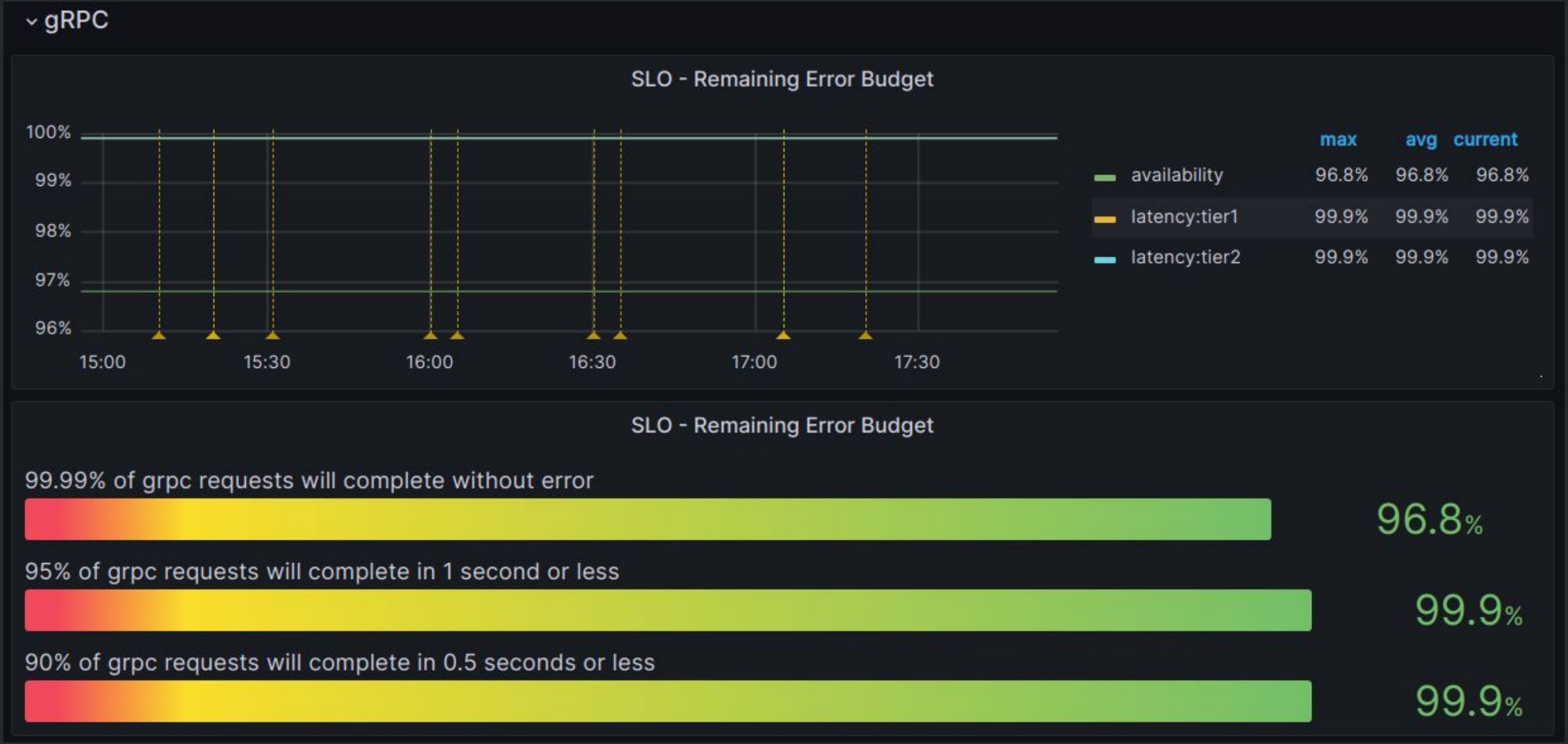
```
groups:
- name: slo-sum.rules
   interval: 1m
   rules:
   - record: slo_calc:http:code:sum
      expr: sum(http_timer_bucket{le="+Inf"}) by (app,code)
   - record: slo_calc:http:time:sum
      expr: sum(http_timer_bucket) by (app,le)
   - record: slo_calc:grpc:code:sum
      expr: sum(grpc_timer_bucket{le="+Inf"}) by (app,code)
   - record: slo_calc:grpc:time:sum
      expr: sum(grpc_timer_bucket) by (app,le)
```

$$\frac{\text{Budgeted} - \text{Actual}}{\text{Budgeted}} = \text{\% Budget Remaining}$$

```
(
   ( sum(increase(slo_calc:http:code:sum{app="feature-flags"}[28d])) * .01 )
   -
   (
     sum(increase(slo_calc:http:code:sum{app="feature-flags"}[28d]))
     -
     sum(increase(slo_calc:http:code:sum{app="feature-flags",code!~"5.."}[28d]))
   )
 )
/
( sum(increase(slo_calc:http:code:sumapp="feature-flags"[28d])) * .01 )
```

Per-App Record Rule

```
groups:
- interval: 3m
  name: feature-flags.slo.rules
  rules:
  - record: slo
    expr: |
         (
            ( sum(increase(slo_calc:http:code:sum{app="feature-flags"}[28d])) * .01 )
            -
            (
               sum(increase(slo_calc:http:code:sum{app="feature-flags"}[28d]))
               -
               sum(increase(slo_calc:http:code:sum{app="feature-flags",code!~"5.."}[28d]))
            )
         )
         /
         (sum(increase(slo_calc:http:code:sum{app="feature-flags"}[28d])) * .01
    labels:
      app: feature-flags
      objective: availability
      description: 99.9% of grpc requests will complete without error
      type: http
```



⌄ gRPC

SLO - Remaining Error Budget

	max	avg	current
🟩 availability	96.8%	96.8%	96.8%
🟨 latency:tier1	99.9%	99.9%	99.9%
🟦 latency:tier2	99.9%	99.9%	99.9%

SLO - Remaining Error Budget

99.99% of grpc requests will complete without error

96.8%

95% of grpc requests will complete in 1 second or less

99.9%

90% of grpc requests will complete in 0.5 seconds or less

99.9%

Attempt 4 Results

Good 👍

- Query Completes for every app all the time!
- No more random spikes!
- Final metrics are fast in dashboard
- Final metrics are easily templated

Attempt 5?

Ok, so…

https://prometheus.io/docs/prometheus/latest/querying/functions/#increase

`increase` should only be used with counters and native histograms where the components behave like counters. [...]

```
(
      n(increase(slo_calc:http:code:sum{app="feature-flags"}[28d])) * .01 )
   -
   (
       (increase(slo_calc:http:code:sum{app="feature-flags"}[28d]))
     -
       (increase(slo_calc:http:code:sum{app="feature-flags",code!~"5.."}[28d]))
   )
 )
/
     (increase(slo_calc:http:code:sumapp="feature-flags"[28d])) * .01 )
```

```
sum(increase(slo_calc:http:code:sum{app="feature-flags"})[28d]))
```

```
  - record: slo_calc:http:code:sum
    expr: sum(http_timer_count{}) by (app,code)
```

```
sum(increase(sum(http_timer_count{app="feature-flags"}) by (app,code))[28d]))
```

$$\frac{\text{Budgeted} - \text{Actual}}{\text{Budgeted}} = \text{\% Budget Remaining}$$

$$1 - \frac{\text{Actual}}{\text{Budgeted}} = \text{\% Budget Remaining}$$

$$\left(\left(\boxed{x} * \boxed{C} \right) - \left(\boxed{x} - \boxed{y} \right) \right) / \left(\boxed{x} * \boxed{C} \right)$$

$$\frac{xC-(x-y)}{xC}$$

$$\frac{xC - (x - y)}{xC}$$

$$\Downarrow$$

$$1 - \left(\frac{1}{C}\right) - \left(\frac{y}{xC}\right)$$

Simplify Proofs

$$\frac{xC-(x-y)}{xC} \Rightarrow \frac{xC}{xC}-\left(\frac{x-y}{xC}\right)$$

$$\Rightarrow 1-\frac{(x-y)}{(xC)} \Rightarrow 1-\left(\frac{x}{xC}\right)-\left(\frac{y}{xC}\right)$$

$$\Rightarrow 1-\left(\frac{1}{1C}\right)-\left(\frac{y}{xC}\right) \Rightarrow 1-\left(\frac{1}{C}\right)-\left(\frac{y}{xC}\right)$$

```
1 - (
 sum(increase(http_timer_count{app="feature-flags",code=~"5.."}[28d]))
 or vector(0)
 /
 (sum(increase(http_timer_count{app="feature-flags"}[28d])) * .01)
)
```

Final Per-App Record Rule

```
groups:
- interval: 3m
  name: feature-flags.slo.rules
  rules:
  - record: slo
    expr: |
         1 - (
           sum(increase(http_timer_count{app="feature-flags",code=~"5.."}[28d]))
           or vector(0)
           /
           (sum(increase(http_timer_count{app="feature-flags"}[28d])) * .01)
         )
    labels:
      app: feature-flags
      objective: availability
      description: 99.99% of grpc requests will complete without error
      type: http
```

What About Scale?

Option: Reduce Range

```
1 - (
 sum(increase(http_timer_count{app="feature-flags",code=~"5.."}[14d]))
 or vector(0)
 /
 (sum(increase(http_timer_count{app="feature-flags"}[14d])) * .01)
)
```

Option: Re-record (without sum)

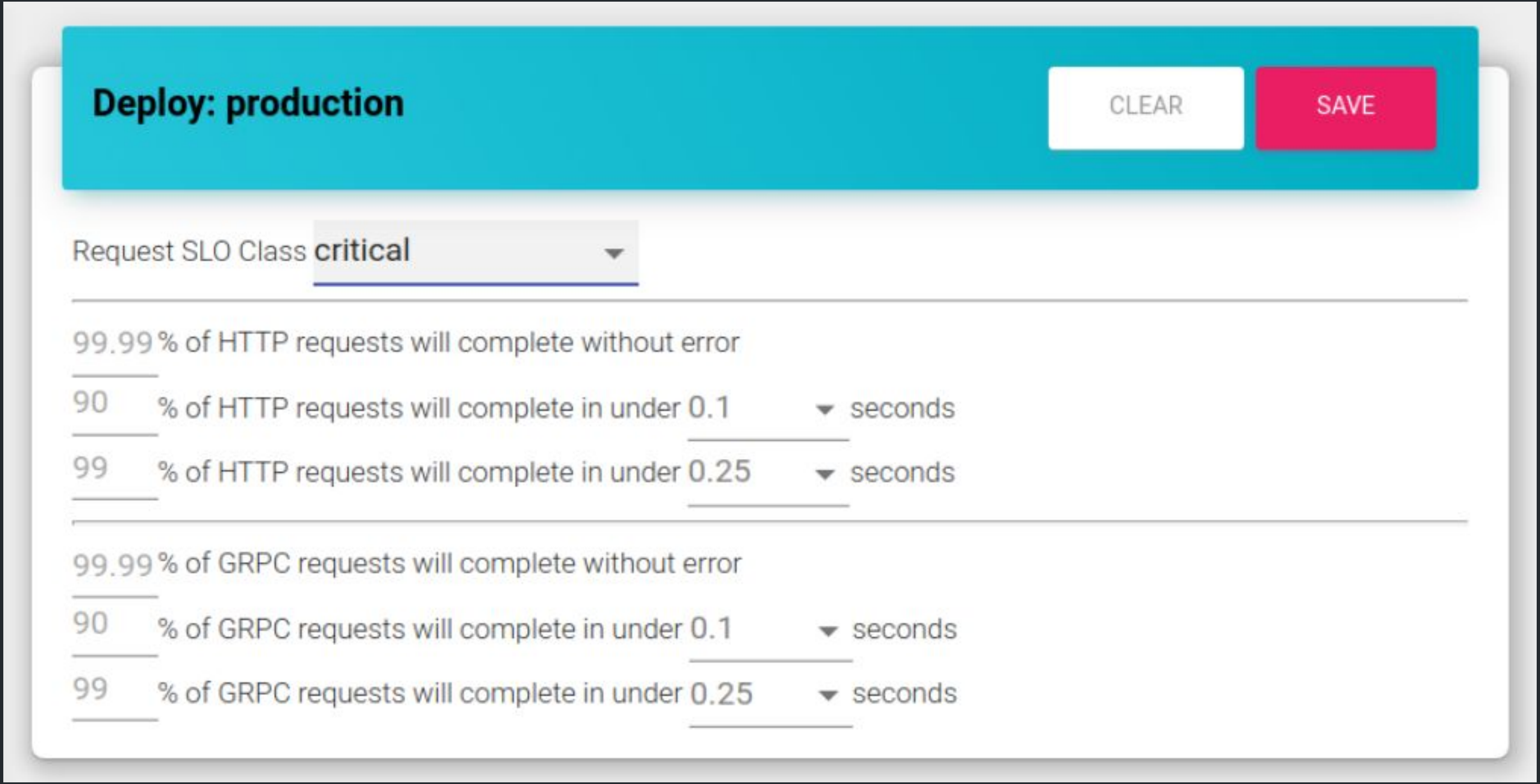
```
groups:
- name: slo-reduce.rules
   interval: 5m
   rules:
   - record: slo_calc:http_timer_count
      expr: http_timer_count
```

No "sum()"!

```
1 - (
 sum(increase(slo_calc:http_timer_count{app="feature-flags",code=~"5.."}[28d]))
 or vector(0)
 /
 (sum(increase(slo_calc:http_timer_count{app="feature-flags"}[28d])) * .01)
)
```

Provider Change

VICTORIA
METRICS



Request SLO Class critical

99.99 % of HTTP requests will complete without error

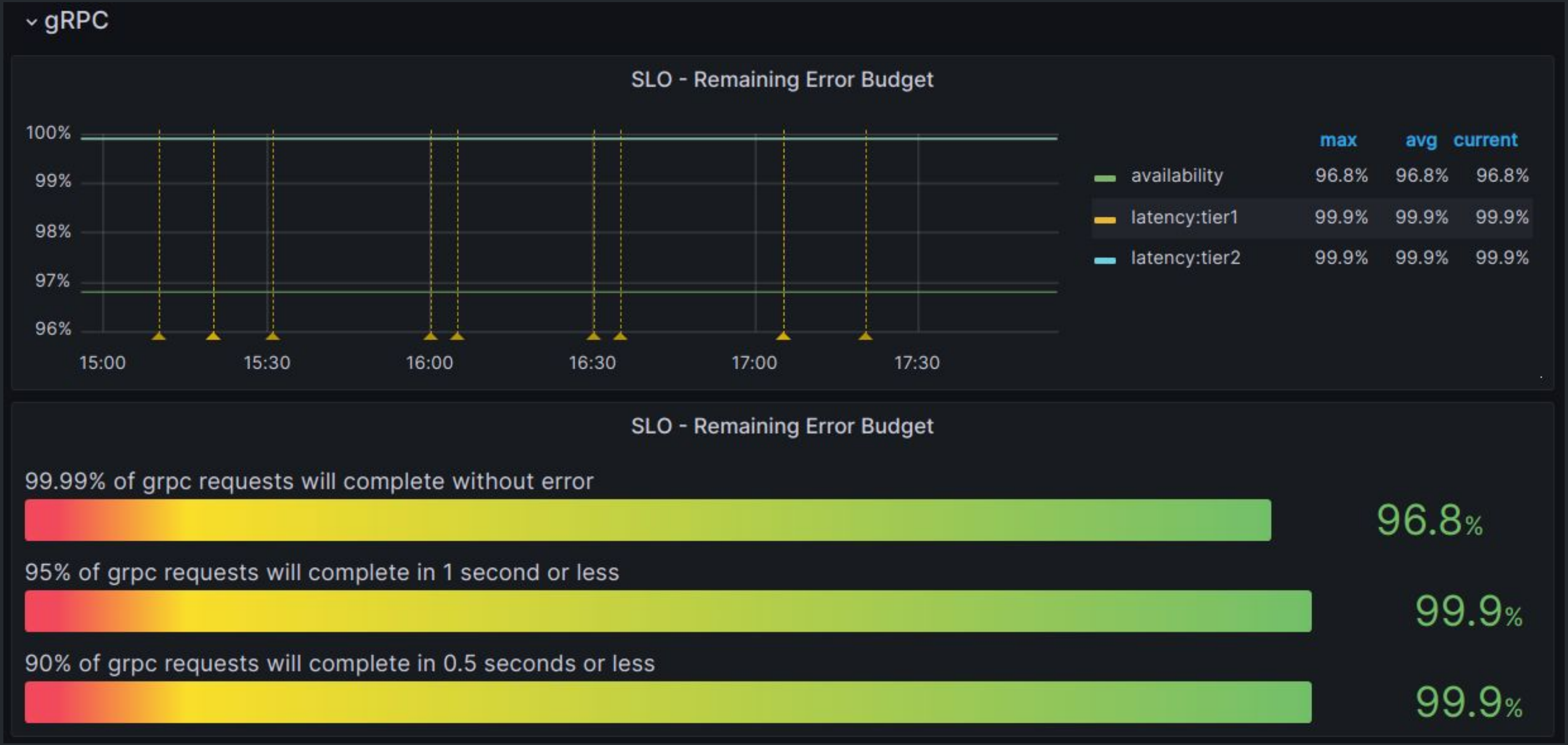
90 % of HTTP requests will complete in under 0.1 seconds

99 % of HTTP requests will complete in under 0.25 seconds

99.99 % of GRPC requests will complete without error

90 % of GRPC requests will complete in under 0.1 seconds

99 % of GRPC requests will complete in under 0.25 seconds



gRPC

SLO - Remaining Error Budget

		max	avg	current
🟩	availability	96.8%	96.8%	96.8%
🟨	latency:tier1	99.9%	99.9%	99.9%
🟦	latency:tier2	99.9%	99.9%	99.9%

SLO - Remaining Error Budget

99.99% of grpc requests will complete without error — 96.8%

95% of grpc requests will complete in 1 second or less — 99.9%

90% of grpc requests will complete in 0.5 seconds or less — 99.9%

Final thoughts

- Keep inputs simple and clean
- Standardize metric names and labels
- Use record rules to get dashboard friendly metrics
  - Be very careful with record rules
  - Be *very, very* careful with them in other queries!
    - Test logic by putting the rule query in the place of the rule

Thank you!

https://engineering.getweave.com/

Carson Anderson

DevX-O, Weave