

Smoothed and Anchored Rate Calculation in PromQL

Björn "Beorn" Rabenstein

KubeCon + CloudNativeCon, Amsterdam, Netherlands. 2026-03-24

rate* – most used PromQL function

`rate(requests_total[1m])`

also most misunderstood, most discussed...

rate yrate arate

xrate hrate

irate



⚙️ * I.e. rate and friends: rate, increase, delta.

What's wrong with the existing rate?

tl;dr: Nothing, but it's always a trade-off. Focus on downsides here.

Properties:

- Strictly looks inside the range. → Hard to get full coverage in graphing.
Needs two samples in the range.
- Extrapolates to range boundaries. → Results are often non-integer.
- Heuristics to guess where a series starts/ends. → Can go wrong.
- Not composable. → $\text{delta}(\text{foo}[1\text{m}]) + \text{delta}(\text{foo}[1\text{m}] \text{ offset } 1\text{m}) \neq \text{delta}(\text{foo}[2\text{m}])$



For ... reasons, things got moving last year...

- Beorn dumped everything he knew into a [giant doc](#).
- Julien distilled it into an actionable [design doc](#)...
- ...and implemented it!

`rate(requests_total[1m] anchored)`

`rate(requests_total[1m] smoothed)`



Design doc (PROM-52)

<https://github.com/prometheus/proposals/blob/main/proposals/0052-extended-range-selectors-semantic.md>

Everything else is linked from there.



Try it out yourself (Prometheus v3.7+):

`--enable-feature=promql-extended-range-selectors`

🤖 What you should actually run:

`--enable-feature=promql-extended-range-selectors,created-timestamp-zero-ingestion`

(Doesn't work with native histograms yet.)

Smoothed/Anchored rate: Support for histograms #17225

Open



roidelapluie opened on Sep 25, 2025

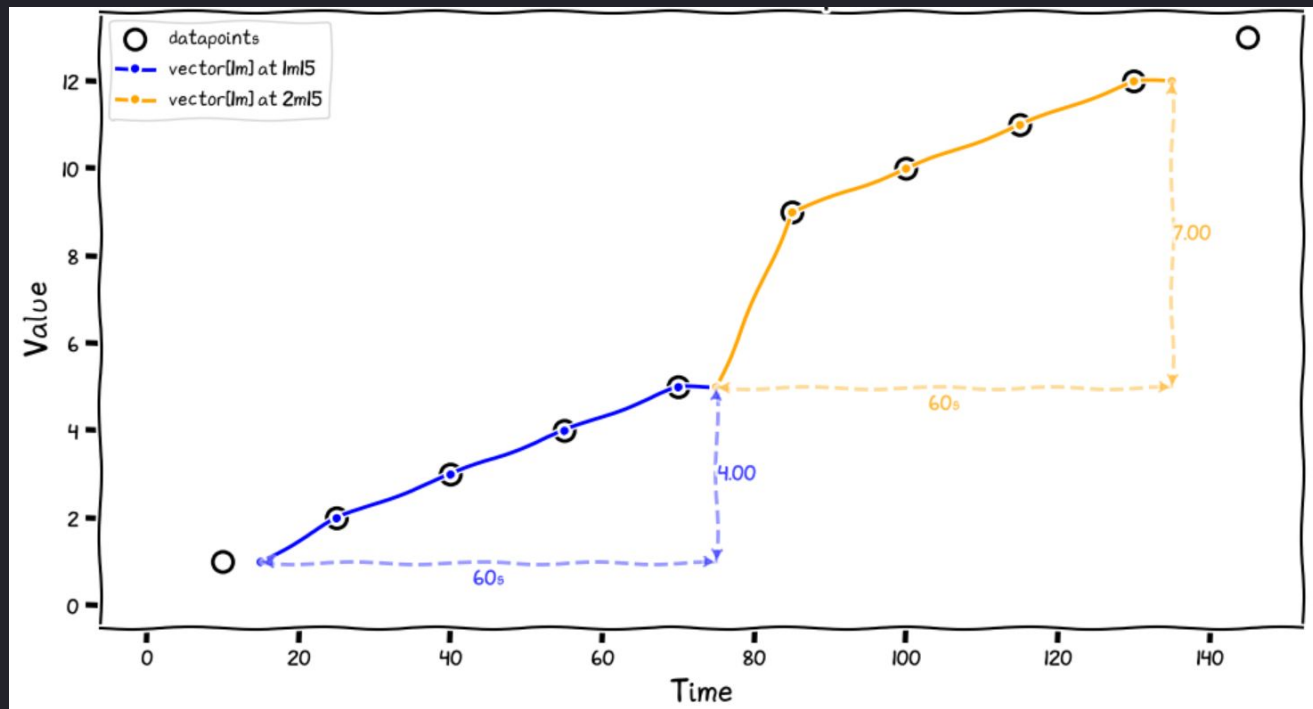
Member

Smoothed and Anchored rate have been implemented behind a feature flag [#16457](#) for *floats*.

This issue tracks the support for Histograms. A design doc might be required to know what to do in case of schema changes and other corner cases.



rate(requests_total[1m] anchored)



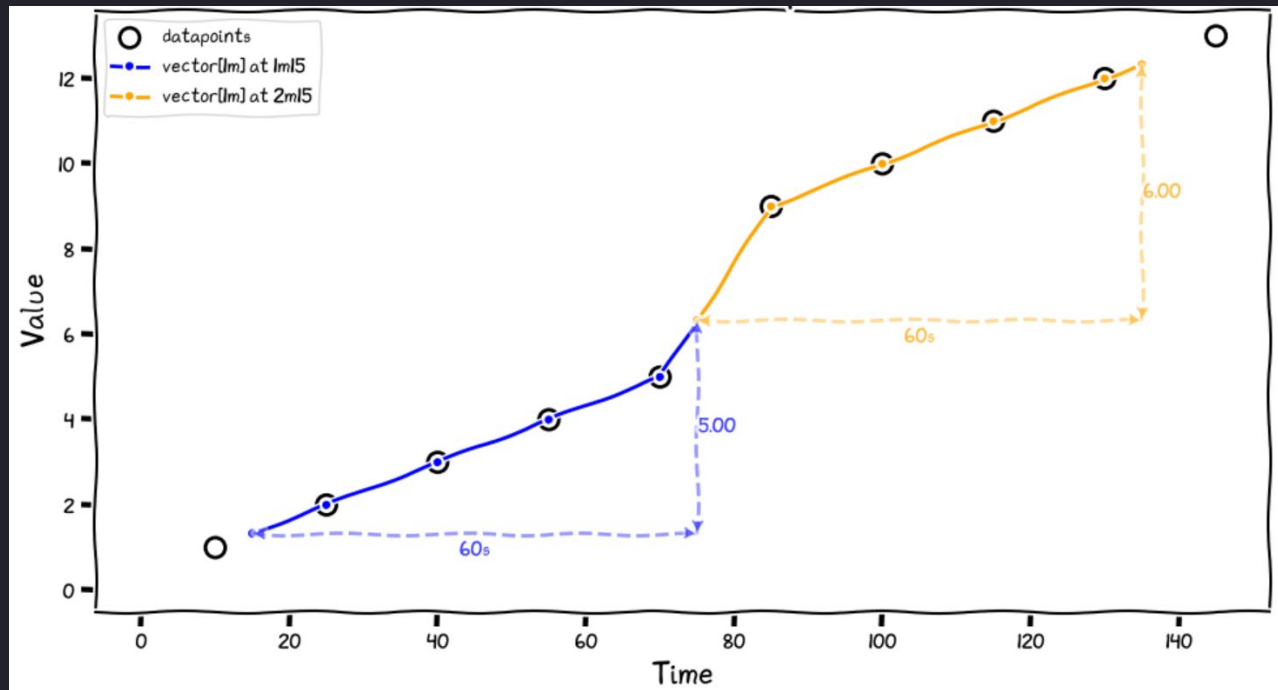
Direct difference
between actual values.

- integers stay integers
- works with single points in the range



$(\text{requests_total} - \text{requests_total offset 1m}) / 1\text{m}$
with counter reset detection but without staleness handling

rate(requests_total[1m] smoothed)



Interpolation at the range boundaries.

- improved estimation
- needs data "from the future"*
- works with zero points in the range ✨



* Use a **query_offset** of one or two scrape intervals in rules, including alerts.

sum by (batch) (rate(queries_total[1m]))



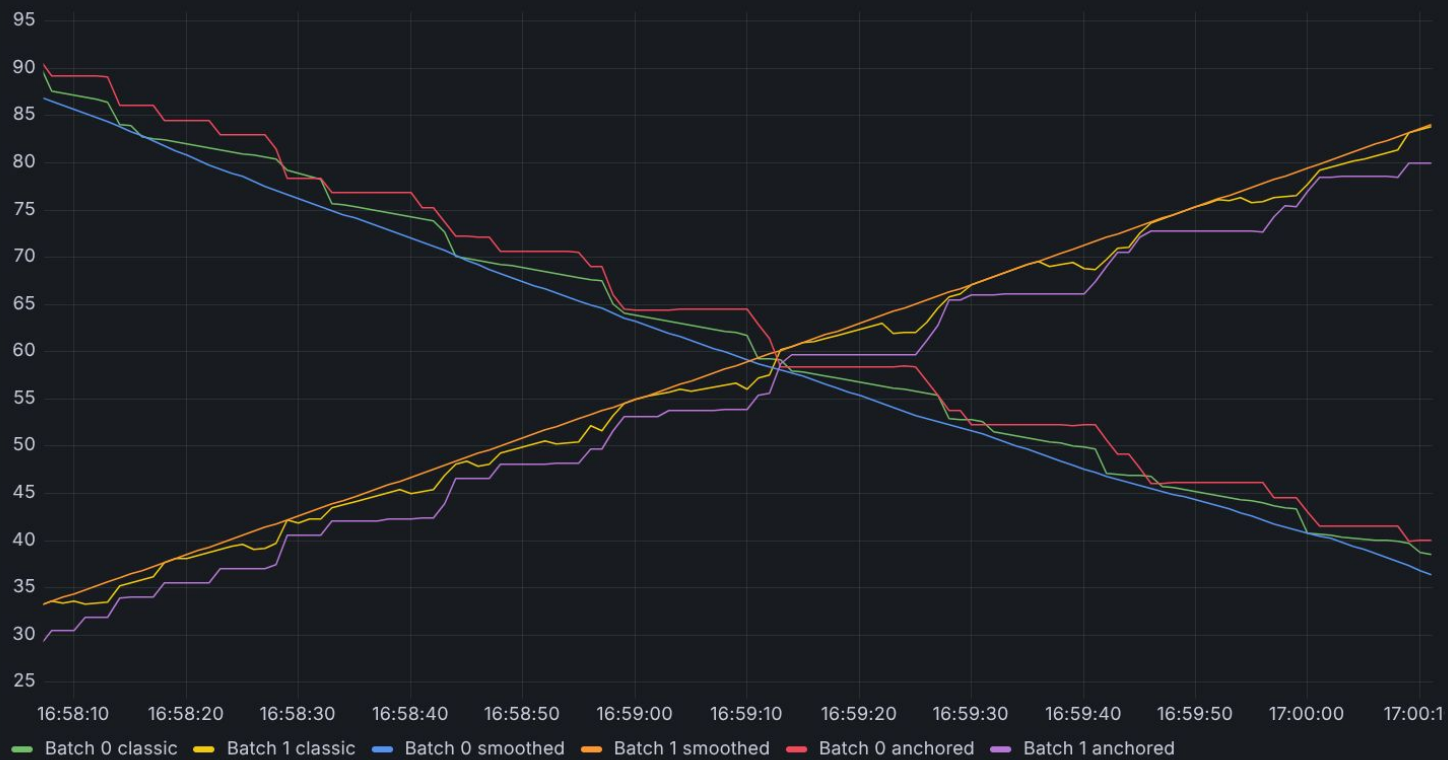
sum by (batch) (rate(queries_total[1m] smoothed))



Rolling restart of a service with 20 workers serving 123.45 qps



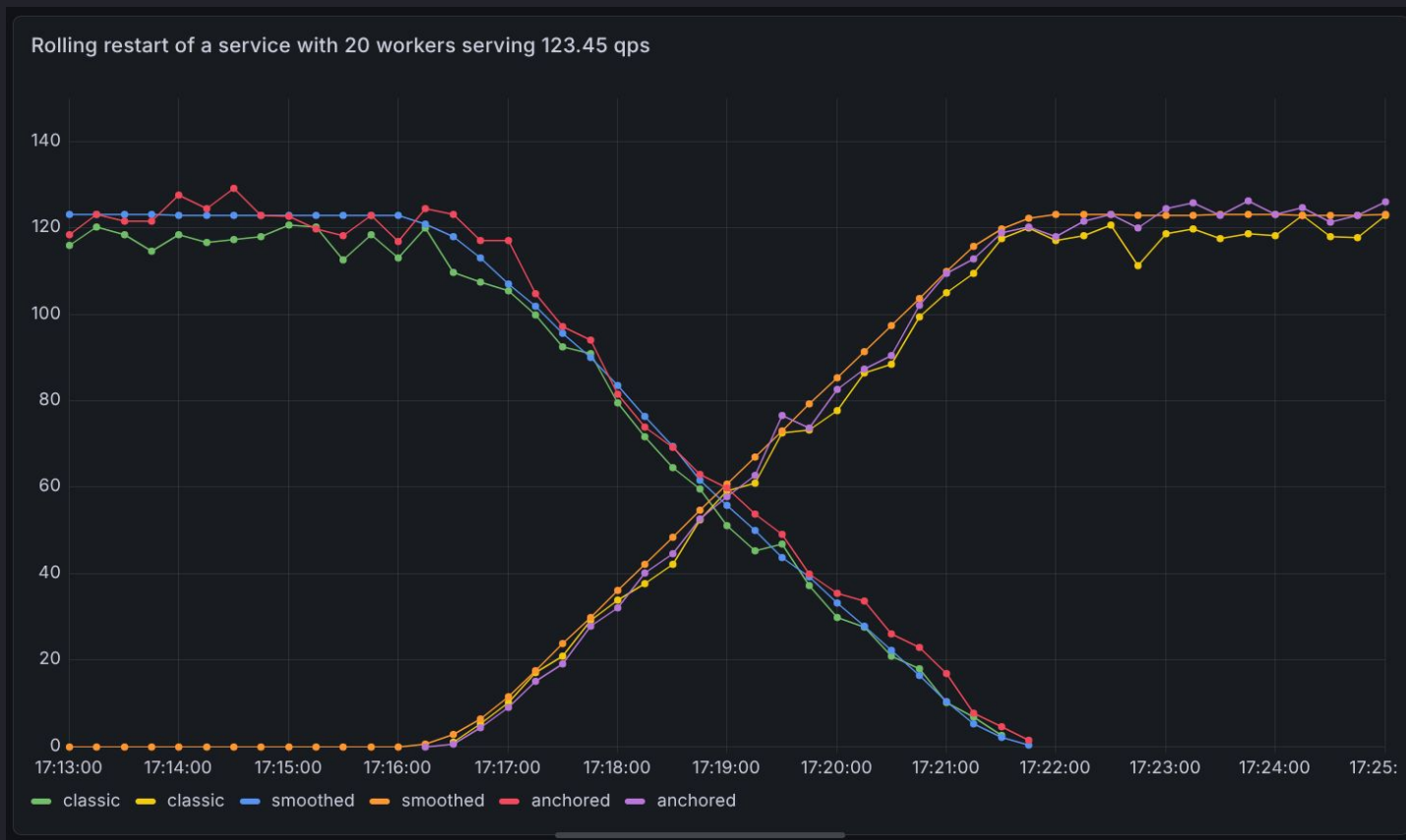
Rolling restart of a service with 20 workers serving 123.45 qps



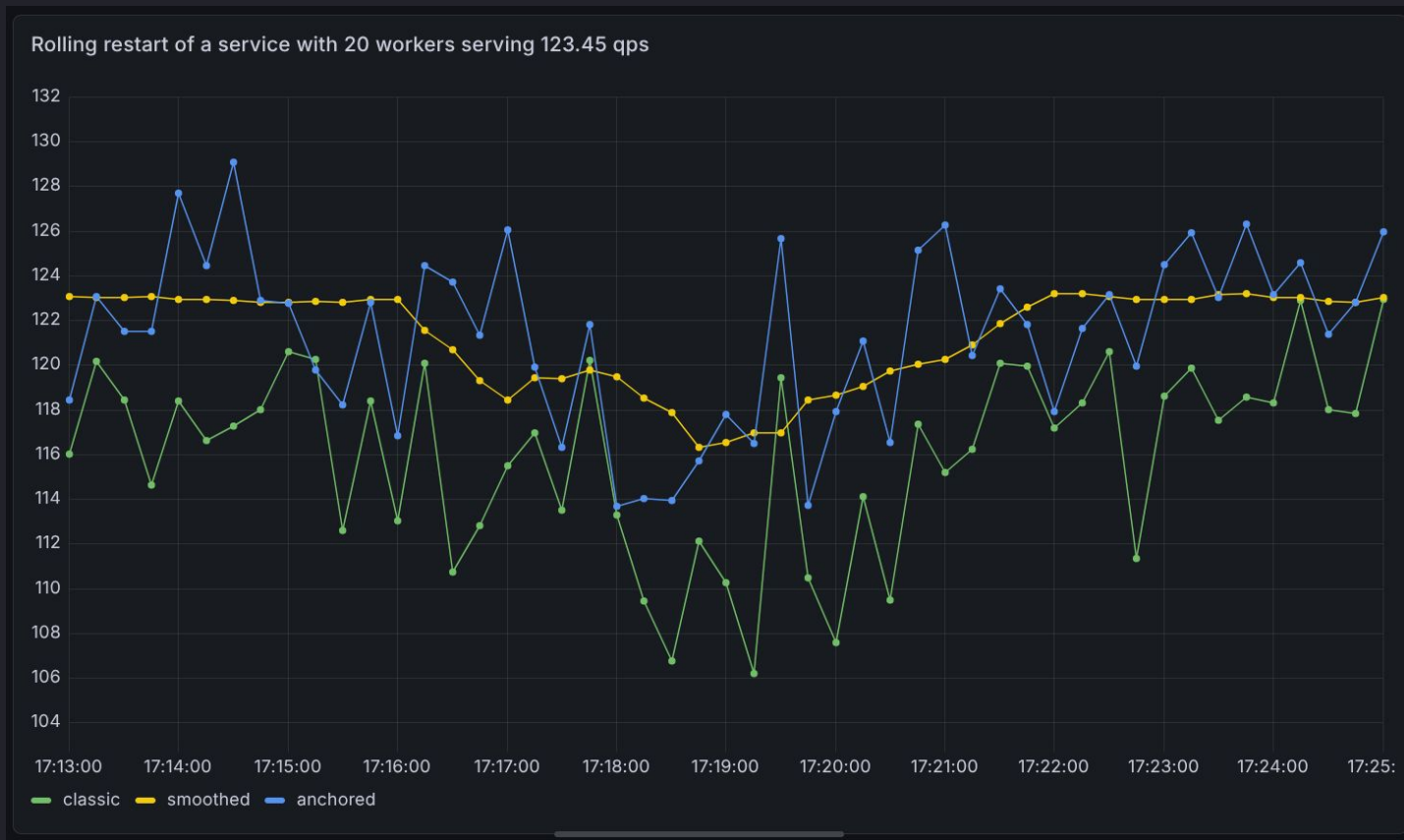
`sum(rate(queries_total[1m]))` (also with `smoothed` and `anchored`)



10% failed scrapes – sum by batch



10% failed scrapes – overall sum



Summary

```
rate(queries_total[step()] smoothed)
(or rate(queries_total[$__interval] smoothed) in Grafana)
```

- `smoothed` and `anchored` both gives you “perfect coverage” in graphing.
- Both are “almost composable”.
- Neither relies on any heuristics about beginning and end of a series.
- `anchored` needs only one sample in the range.
- `smoothed` even works with no samples in the range.
- `smoothed` underestimates the rate for “now”. Needs a query offset.
- `anchored` gives you the integer result you wanted, for better or worse...
- `anchored` introduces a variable delay (depending on sample alignment).
- `smoothed` handles multiple workers and lost scrapes most gracefully.



Why did it take so long?

"Just add them as xrate, yrate, ... It's not even a breaking change."

"No is temporary, yes is forever."

"Mark it as experimental."

"Users won't notice and rely on it in critical paths, and then we cannot take it away."

"Hide it behind a feature flag."

"It's a core feature and already quite confusing. Imagine the confusion if we added a handful of half-baked alternatives. It's like... Go generics!"



What changed last year?

- Start timestamp* (formerly known as created timestamp)
- PromQL is used in many places with ingestion delay
- Prometheus v3 broke a common work-around

v3.0.0: [CHANGE] PromQL: Range selectors and the lookback delta are now left-open, i.e. a sample coinciding with the lower time limit is excluded rather than included.

v3.0.1: [BUGFIX] Promql: Make subqueries left open.

* Needs to be supported by instrumentation and to be enabled on the Prometheus side, see

https://prometheus.io/docs/prometheus/latest/feature_flags/#start-created-timestamps-zero-injection





Grafana Labs

<https://github.com/beorn7/talks>