Prometheus PromQL and promtool

START HERE

`up`
Which targets are scraped

`http_requests_total{job="api"}`
Select series by label

`rate(<counter>[5m])`
Per-second counter rate

`sum by (<label>) (<expr>)`
Aggregate and group

`histogram_quantile(0.95, sum by (le) (rate(<m>_bucket[5m])))`
p95 latency

`promtool check config prometheus.yml`
Validate config

`promtool test rules <tests.yml>`
Unit-test rules

`curl -X POST localhost:9090/-/reload`
Reload config

Selector http_requests_total{...} → `[5m]` → **Range vector** samples per series → `rate()` → **Instant vector** one value per series → `sum by` → **Aggregated** fewer series

CONCEPTS

An instant query returns one value per series at a single time; a range query evaluates at each step. Selectors give instant vectors, a [5m] suffix gives a range vector, and rate() turns a range vector back into an instant vector.

Counters only go up: always wrap them in rate() or increase() before graphing or summing. Use the plain value only for gauges.

Matchers are fully anchored regexes, so =~"foo" matches only foo. Use =~"foo.*" for prefixes.

SELECTORS

`<metric>`
Instant vector of all series

`<metric>{<label>="<v>"}`
Exact label match

`<metric>{<label>≠"<v>"}`
Label not equal

`<metric>{<label>=~"<re>"}`
Regex match

`<metric>{<label>!~"<re>"}`
Regex non-match

SELECTORS (cont.)

`<metric>[5m]`
Range vector over 5 minutes
Left edge is exclusive, right edge inclusive.

`<metric> offset 1h`
Value from one hour ago
offset must directly follow the selector, not an aggregation.

`<expr>[30m:1m]`
Subquery: expr every 1m for 30m

Durations: ms s m h d w y, combinable in descending order, e.g. 1h30m. A float cannot be combined with a unit.

COUNTERS AND GAUGES

`rate(<counter>[5m])`
Per-second average rate
Pick a window of at least four scrape intervals.

`irate(<counter>[5m])`
Rate from the last two samples

`increase(<counter>[1h])`
Total increase over the window

`changes(<gauge>[1h])`
How often the value changed

`absent(<metric>{<label>="<v>"})`
1 when no series exist

AGGREGATION

`sum(<expr>)`
Total across all series

`sum by (<label>) (<expr>)`
Sum, keeping only these labels

`sum without (<label>) (<expr>)`
Sum, dropping these labels

`avg by (<label>) (<expr>)`
Average per group

`max by (<label>) (<expr>)`
Maximum per group

`count by (<label>) (<expr>)`
Number of series per group

`topk(5, <expr>)`
Five largest series

OVER TIME AND HISTOGRAMS

`avg_over_time(<gauge>[1h])`
Average per series over the range

`max_over_time(<gauge>[1h])`
Maximum per series over the range

`histogram_quantile(0.95, rate(<m>_bucket[5m]))`
Quantile from buckets
Keep the le label in any sum by (...) before the call.

OPERATORS AND LABELS

`<a> / <b>`
Divide series with identical labels

`<a> / on(<label>) <b>`
Match only on these labels

`<a> / ignoring(<label>) <b>`
Match ignoring these labels

`<a> * on(<l>) group_left(<x>) <b>`
Many-to-one join, copy label x

`<expr> > 100`
Keep series above a threshold

`<a> and <b>`
Series of a that also appear in b

`label_replace(<expr>, "dst", "$1", "src", "(.*):.*")`
Derive a label via regex

Binary operators drop the metric name. Without on() or ignoring(), both sides need identical label sets to pair.

RUNNING THE SERVER

`prometheus --config.file=<file>`
Start with a config file

RUNNING THE SERVER (cont.)

`prometheus --storage.tsdb.path=<dir>`
Where the TSDB lives

`prometheus --web.enable-lifecycle`
Allow HTTP reload and quit

`curl -X POST <host>:9090/-/reload`
Reload the config

PROMTOOL CHECK

`promtool check config <file>`
Validate config and referenced rule files

`promtool check rules <rules.yml>`
Validate rule files

PROMTOOL QUERY AND TEST

`promtool query instant <url> '<expr>'`
Run an instant query

`promtool query range <url> '<expr>' --start=<t> --end=<t> --step=1m`
Run a range query

`promtool test rules <tests.yml>`
Run rule unit tests

PROMTOOL TSDB AND PUSH

`promtool tsdb analyze <dir>`
Cardinality and churn report

RECIPES

Error ratio per service

```
sum by (service) (rate(
http_requests_total{code=~"5..
"}[5m]))
  / sum by (service)
(rate(http_requests_total[5m]))
```

Latency p95 by route

```
histogram_quantile(0.95,
  sum by (le)
(rate(<m>_bucket[5m])))
```

Rule file with a unit test

```
# rules.yml
groups:
- name: demo
  rules:
  - alert: Down
    expr: up == 0
    for: 5m
# tests.yml
rule_files: [rules.yml]
tests:
- interval: 1m
  input_series:
  - series: 'up{job="a"}'
    values: '0x10'
  alert_rule_test:
  - eval_time: 10m
    alertname: Down
```