PromQL Cheat Sheet

Want to learn PromQL from the ground up? Check out our self-paced in-depth PromQL training.

Selecting series

Select latest sample for series with a given metric name:

```
node_cpu_seconds_total
```

Open in PromLens

Select 5-minute range of samples for series with a given metric name:

```
node_cpu_seconds_total[5m]
```

Open in PromLens

Only series with given label values:

```
node_cpu_seconds_total{cpu="0",mode="idle"}
```

Open in PromLens

Complex label matchers (`=`: equality, `!=`: non-equality, `=~`: regex match, `!~`: negative regex match):

```
node_cpu_seconds_total{cpu!="0",mode=~"user|system"}
```

Open in PromLens

Select data from one day ago and shift it to the current time:

```
process_resident_memory_bytes offset 1d
```

Open in PromLens

Rates of increase for counters

Per-second rate of increase, averaged over last 5 minutes:

```
rate(demo_api_request_duration_seconds_count[5m])
```

Open in PromLens

Per-second rate of increase, calculated over last two samples in a 1-minute time window:

```
irate(demo_api_request_duration_seconds_count[1m])
```

Open in PromLens

Absolute increase over last hour:

```
increase(demo_api_request_duration_seconds_count[1h])
```

Open in PromLens

Aggregating over multiple series

Sum over all series:

```
sum(node_filesystem_size_bytes)
```

Open in PromLens

Preserve the `instance` and `job` label dimensions:

```
sum by(job, instance) (node_filesystem_size_bytes)
```

Open in PromLens

Aggregate away the `instance` and `job` label dimensions:

```
sum without(instance, job) (node_filesystem_size_bytes)
```

Open in PromLens

Available aggregation operators: `sum()`, `min()`, `max()`, `avg()`, `stddev()`, `stdvar()`, `count()`, `count_values()`, `group()`, `bottomk()`, `topk()`, `quantile()`

Math between series

Add all equally-labelled series from both sides:

```
node_memory_MemFree_bytes + node_memory_Cached_bytes
```

Open in PromLens

Add series, matching only on the `instance` and `job` labels:

```
node_memory_MemFree_bytes + on(instance, job) node_memory_Cached_bytes
```

Open in PromLens

Add series, ignoring the `instance` and `job` labels for matching:

```
node_memory_MemFree_bytes + ignoring(instance, job) node_memory_Cached_byte
```

Open in PromLens

Explicitly allow many-to-one matching:

```
rate(demo_cpu_usage_seconds_total[1m]) / on(instance, job) group_left demo_
```

Open in PromLens

Include the `version` label from "one" (right) side in the result:

```
node_filesystem_avail_bytes * on(instance, job) group_left(version) node_ex
```

Open in PromLens

Available arithmetic operators: `+`, `-`, `*`, `/`, `%`, `^`

Filtering series by value

Only keep series with a sample value greater than a given number:

```
node_filesystem_avail_bytes > 10*1024*1024
```

Open in PromLens

Only keep series from the left-hand side whose sample values are larger than their right-hand-side matches:

```
go_goroutines > go_threads
```

Open in PromLens

Instead of filtering, return `0` or `1` for each compared series:

```
go_goroutines > bool go_threads
```

Open in PromLens

Match only on specific labels:

```
go_goroutines > bool on(job, instance) go_threads
```

Open in PromLens

Available comparison operators: `==`, `!=`, `>`, `<`, `>=`,`<=`

Set operations

Include any label sets that are either on the left or right side:

```
up{job="prometheus"} or up{job="node"}
```

Open in PromLens

Include any label sets that are present both on the left and right side:

```
node_network_mtu_bytes and (node_network_address_assign_type == 0)
```

Open in PromLens

Include any label sets from the left side that are not present in the right side:

```
node_network_mtu_bytes unless (node_network_address_assign_type == 1)
```

Open in PromLens

Match only on specific labels:

```
node_network_mtu_bytes and on(device) (node_network_address_assign_type ==
```

Open in PromLens

Quantiles from histograms

90th percentile request latency over last 5 minutes, for every label dimension:

```
histogram_quantile(0.9, rate(demo_api_request_duration_seconds_bucket[5m]))
```

Open in PromLens

...for only the `path` and `method` dimensions:

```
histogram_quantile(
  0.9,
  sum by(le, path, method) (
    rate(demo_api_request_duration_seconds_bucket[5m])
  )
)
```

Open in PromLens

Changes in gauges

Per-second derivative using linear regression:

```
deriv(demo_disk_usage_bytes[1h])
```

Open in PromLens

Absolute change in value over last hour:

```
delta(demo_disk_usage_bytes[1h])
```

Open in PromLens

Predict value in 1 hour, based on last 4 hours:

```
predict_linear(demo_disk_usage_bytes[4h], 3600)
```

Open in PromLens

Aggregating over time

Average within each series over a 5-minute period:

```
avg_over_time(go_goroutines[5m])
```

Open in PromLens

Get the maximum for each series over a one-day period:

```
max_over_time(process_resident_memory_bytes[1d])
```

Open in PromLens

Count the number of samples for each series over a 5-minute period:

```
count_over_time(process_resident_memory_bytes[5m])
```

Open in PromLens

See all available `xxx_over_time()` aggregation functions.

Time

Get the Unix time in seconds at each resolution step:

```
time()
```

Open in PromLens

Get the age of the last successful batch job run:

```
time() - demo_batch_last_success_timestamp_seconds
```

Open in PromLens

Find batch jobs which haven't succeeded in an hour:

```
time() - demo_batch_last_success_timestamp_seconds > 3600
```

Open in PromLens

Dealing with missing data

Create one output series when the input vector is empty:

```
absent(up{job="some-job"})
```

Open in PromLens

Create one output series when the input range vector is empty for 5 minutes:

```
absent_over_time(up{job="some-job"}[5m])
```

Open in PromLens

Manipulating labels

Join the values of two labels with a `-` separator into a new `endpoint` label:

```
label_join(rate(demo_api_request_duration_seconds_count[5m]), "endpoint", "
```

Open in PromLens

Extract part of a label and store it in a new label:

```
label_replace(up, "hostname", "$1", "instance", "(.+):(\\d+)")
```

Open in PromLens

Subqueries

Calculate the 5-minute-averaged rate over a 1-hour period, at the default subquery resolution (= global rule evaluation interval):

```
rate(demo_api_request_duration_seconds_count[5m])[1h:]
```

Open in PromLens

Calculate the 5-minute-averaged rate over a 1-hour period, at a 15-second subquery resolution:

```
rate(demo_api_request_duration_seconds_count[5m])[1h:15s]
```

Open in PromLens

Using the subquery result to get the maximum rate over a 1-hour period:

```
max_over_time(
  rate(
    demo_api_request_duration_seconds_count[5m]
  )[1h:]
)
```

Open in PromLens

More...

For more details about PromQL, see the official PromQL documentation:

- Basics
- Operators
- Functions
- Examples

Want to learn more? Check out our self-paced in-depth PromQL training by the c