

# PromQL

- PromQL Übersicht
- Beispiele
  - Windows CPU-Auslastung anzeigen

# PromQL Übersicht

## Auswählen von Metriken

Letzten Datensatz anhand des Metrik-Namens auswählen:

```
node_cpu_seconds_total
```

Letzte Datensätze innerhalb eines Zeitraums auswählen:

```
node_cpu_seconds_total[5m]
```

Datensätze mit bestimmten "Label" abrufen:

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

Datensätze mit komplexer Bezeichnungsübereinstimmung abrufen:

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

=	Gleich
!=	Nicht gleich
=~	Regex Übereinstimmung
!~	Negativer Regex Übereinstimmung

Wählen Sie Daten von vor einer Zeitspanne aus und verschieben Sie die Daten auf die aktuelle Zeit:

```
process_resident_memory_bytes offset 1d
```

## Rechnen zwischen verschiedenen Metriken

Addiere zwei Metriken mit demselben "Label" miteinander:

```
node_memory_MemFree_bytes + node_memory_Cached_bytes
```

Addiere Metriken miteinander, die nur mit dem `instance` und dem `job` "Label" übereinstimmen:

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

Addiere Metriken miteinander, wobei das `instance` und `job` "Label" ignoriert werden:

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

Mehrfachabgleich explizit erlauben:

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

Versionsbezeichnung in die rechte Seite des Ergebnisses hinzufügen:

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

**Verfügbare Arithmetische Operatoren:** +, -, \*, /, %, ^

## Quantile aus Histogrammen

90. Perzentil der Anfragelatenz der letzten 5 Minuten für jede Etikettendimension:

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

... nur für die Pfad- und Methodendimensionen:

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

## Uhrzeit

Abfrage für die UNIX-Zeit in Sekunden bei jedem Schritt:

```
time()
```

Ermittelt das Alter des zuletzt laufenden Batch-Jobs:

```
time() - demo_batch_last_success_timestamp_seconds
```

Finden Sie Batch-Aufträge, die in einer Stunde nicht erfolgreich waren:

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

## Unterabfragen

Berechnen Sie den 5-Minuten Durchschnitt über einen Zeitraum von 1 Stunde bei einer Standardauflösung der Unterabfrage (=globales Regelauswertungsintervall)

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

Berechnen Sie den 5-Minuten Durchschnitt über einen Zeitraum von 1 Stunde mit einer Auflösung von 15 Sekunden:

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

## Steigerungsraten für Zähler

Anstiegsrate pro Sekunde, ermittelt über die letzten 5 Minuten:

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

Anstiegsrate pro Sekunde, berechnet über die letzten beiden Messwerte in einem 1-Minuten Zeitfenster:

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

Absoluter Anstieg in der letzten Stunde:

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

## Filtern von Metriken nach Werten

Auswählen von Datensätzen, die höher als eine angegebene Zahl ist:

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

Behalten Sie nur die Reihen auf der linken Seite, deren Werte größer sind als die der rechten Seite:

```
go_goroutines > go_threads
```

Anstatt zu filtern, geben Sie für jede vergleichende Reihe eine ☐ oder eine ☐ zurück:

```
go_goroutines > bool go_threads
```

Übereinstimmung nur bei bestimmten "Labeln":

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

**Verfügbare Vergleichsoperatoren: ==, !=, >, <, >=, <=**

## Änderungen der Messergebnisse

Ableitung pro Sekunde mittels linearer Regression:

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

Absolute Wertänderung in der letzten Stunde:

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

Prognostizieren Sie den Wert in 1 Stunde, basierend auf den letzten 4 Stunden:

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

## Umgang mit fehlenden Daten

Erstellt eine Ausgabeserie, wenn der Eingabevektor leer ist:

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

Erstellt eine Ausgabeserie, wenn der Eingabevektor 5 Minuten lang leer ist:

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

## Aggregieren über mehrere Serien

Summe über alle Serien:

```
sum(node_filesystem_size_bytes)
```

Behalten Sie die Abmessungen der `Instanz` und des `Job` "Labels" bei:

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

Aggregieren Sie die Instanz und Job "Label" weg:

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

**Verfügbare Aggregierungs-Operatoren:** `sum()`, `min()`, `max()`, `avg()`, `stddev()`, `stdvar()`, `count()`, `count_values()`, `group()`, `bottomk()`, `topk()`, `quantile()`

## Eingestellte Operationen

Schließen Sie alle "Label" Datensätze ein, die sich entweder auf der linken oder auf der rechten Seite befinden:

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

Schließen Sie alle "Label" Datensätze ein, die sowohl auf der linken als auch auf der rechten Seite vorhanden sind:

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

Schließen Sie alle "Label" Datensätze der linken Seite ein, die auf der rechten Seite nicht vorhanden sind:

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

Übereinstimmung nur bei bestimmten "Labeln":

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

## Aggregieren über die Zeit

Durchschnitt innerhalb einer Metrik über einen Zeitraum von 5-Minuten:

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

Ermitteln Sie das Maximum für eine Metrik innerhalb eines Tages:

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

Zählen Sie die Anzahl der Datensätze für jede Metrik über einen Zeitraum von 5-Minuten:

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

**Siehe alle verfügbaren `xxx_over_time()` Aggregationsfunktionen:** [Zeit Aggregationsfunktionen](#)

## Manipulation von Labels

Verbinden Sie die Datensätze von zwei Etiketten mit einem "-" Trennzeichen zu einem neuen Endpunkt-"Label":

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

Extrahieren Sie einen Teil eines Labels und speichern Sie diesen in einem neuen "Label":

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

# Beispiele



# Windows CPU-Auslastung anzeigen

```
100 - (avg(rate(windows_cpu_time_total{mode="idle",job="<Job>"}[5m]))) * 100
```