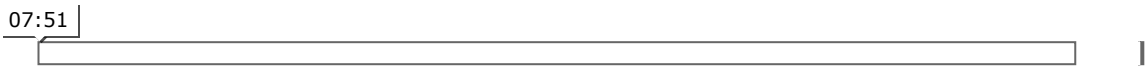
Watch: 🔍 **PromQL**



Prometheus proporciona un lenguaje de consulta funcional llamado **PromQL**(Prometheus Query Language) que permite al usuario seleccionar y agregar datos de series de tiempo en tiempo real. El resultado de una expresión puede mostrarse como un gráfico, verse como datos tabulares en el navegador de expresiones de Prometheus o ser consumido por sistemas externos a través de la API HTTP.

Tipos de datos

En el lenguaje de expresión de Prometheus, una expresión o subexpresión puede evaluar uno de cuatro tipos:

- **Vector instantáneo**: un conjunto de series de tiempo que contiene una sola muestra para cada serie de tiempo, todas compartiendo la misma marca de tiempo

- **Vector de rango**: un conjunto de series de tiempo que contiene un rango de puntos de datos a lo largo del tiempo para cada serie de tiempo

- **Escalar**: un valor numérico simple de coma flotante

Algunos Ejemplos básicos

Devuelve todas las series de tiempo con la métrica http_requests_total:

```
http_requests_total
```

Devuelve todas las series de tiempo con la métrica http_requests_total y las etiquetas de trabajo y controlador dadas:

```
http_requests_total {job = "apiserver", handler = "/ api
/ comments"}
```

Devuelve un rango de tiempo completo (en este caso 5 minutos) para el mismo vector, convirtiéndolo en un vector de rango:

```
http_requests_total {job = "apiserver", handler = "/ api
/ comments"}
```

Usando expresiones regulares, puedes seleccionar series de tiempo solo para trabajos cuyo nombre coincida con un patrón determinado, en este caso, todos los trabajos que finalizan con el servidor:

```
http_requests_total {job = ~ ". * server"}
```

Para seleccionar todos los códigos de estado HTTP excepto los 4xx, puede ejecutar:

```
http_requests_total {status! ~ "4 .."}
```

rate() la función mágica

Sin duda, la función `rate()` o *ratio* será, probablemente, la función más utilizada por métricas de tipo *count*. De hecho no tiene sentido utilizarla con métricas de tipo *gauge*. `rate()` indica el número de incrementos que se han producido en un *count* por segundo. Es una función mágica, porque aunque el *counter* haya sido reiniciado (recuerda que cuando el servicio se reinicia, el *counter* vuelve a empezar desde 0), `rate()` no lo tiene en cuenta y no se alteran los cálculos. Es decir, si el *time series* del *counter* ha evolucionado así: 1, 2, 3, 4, 0, 1, 2 (ha habido un reinicio), `rate()` lo detectará y lo sustituirá por 1, 2, 3, 4, 5, 6, 7.

Funciones de agregación y agrupación

PromQL te permite agregar y agrupar *time series*. Se agrupa gracias a las *labels* y se agrega después gracias a diversas funciones. Por ejemplo, la siguiente *query* devolverá una suma del tráfico recibido por instancia en un periodo de 5 minutos (sumará el valor de todas las diferentes *times series* generadas, recuerda que puede haber más *labels* involucradas).

```
sum(rate(node_network_receive_bytes_total[1m])) by instance
```

Hay que tener cuidado con las funciones de agregación, como *sum*. Se debe tener bien presente que si hacemos *sum* con un *counter* este puede haberse reiniciado y la suma de esos valores no tendrá mucho sentido. Por eso es importante utilizar la funcion `rate()` en compañía de otras funciones de agregación, especialmente con *counters*

Patrones comunes para queries en PromQL

- Gauges:

```
sum(my_gauge)
avg(my_gauge)
max(my_gauge)
min(my_gauge)
```

- Counters:

```
avg(rate(my_counter_total[5m]))
```

- Histogram

```
histogram_quantile(
    0.9,
    sum(rate(my_histogram_latency_seconds_bucket[5m])))
```