



mastery program

> DevOps

Monitoreo con Prometheus

> Chuy Lerma

monitoreo de aplicaciones

- > **metricas**
- > **tipos de metricas**
- > **instrumentación**
- > **etiquetas**



monitoreo de aplicaciones

> métricas



Métricas

- > Time series = metric
- > Una métrica es una combinación de nombre de métrica y su dimensión.
- > Donde la dimensión se diferencia con la etiqueta y su valor.

Métricas

> una métrica `http_total_requests`
> sus dimensiones/etiquetas
`method="POST" y status_code="200"`

Métricas

```
> http_total_requests{method="POST",  
status_code="200"} 1  
> go_goroutines 9
```


Tipos de Métricas

- > Counter
- > Gauge
- > Histogram
- > Summary

monitoreo de aplicaciones

> instrumentación



Instrumentación

- > Lo que haces mayormente.
- > Existen varios clientes.
- > Usaremos Python3



Counter

> Se usan para contar cosas.



Counter

- > Se usan más frecuentemente que otros.
- > Se incrementan por cualquier valor flotante no-negativo.
- > No se decrementan. Regresan a 0



Counter

> Un counter que puede decrementarse sería un Gauge.



Gauge

- > Snapshots del estado actual.
- > Los valores pueden ir para arriba o para abajo.



Gauge

> Cuando quieres monitorear el valor actual de una metrica que puede decrementar a lo largo del tiempo



Gauge

- > No se pueden usar para todo.
- > No se usan si quieres ver la evolución de las métricas.



Gauge Ejemplos

- > el número de elementos en una cola.
- > el uso de memoria de un cache.
- > el número de hilos activos.



Gauge Ejemplos

- > la última vez que un registro fue procesado.
- > el promedio de peticiones por segundo en el último minuto.



Summary

- > es la combinación de otros tipos
- > dos counters y algunos gauges

<http://localhost:9090/metrics>



Summary

`prometheus_rule_evaluation_duration_seconds{quantile="0.5"}
0.000946612`

`prometheus_rule_evaluation_duration_seconds{quantile="0.9"}
0.001950352`

`prometheus_rule_evaluation_duration_seconds{quantile="0.99"}
0.00366201`

`prometheus_rule_evaluation_duration_seconds_sum 5.680487416000003
prometheus_rule_evaluation_duration_seconds_count 5154`

Summary

`_sum` y `_count` casi siempre estaran presentes. Ambos son counters.
`_count` se incrementa en cada `observe`
`_sum` se incrementa por cada valor observado

En nuestro ejemplo se han observado 5154
tambien se observan quantiles.



Summary

- > la última vez que un registro fue procesado.
- > Se usa generalmente para medir la latencia de tu aplicación.



Histogram

- > Es muy similar al Summary
- > Se usan para dar seguimiento a que tanto (en tiempo) un evento toma.



Histogram

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="0.1"}  
25547
```

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="0.2"}  
26688
```

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="0.4"}  
27760
```

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="1"} 28641
```

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="3"} 28782
```

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="8"} 28844
```

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="20"}  
28855
```

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="60"}  
28860
```

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="120"}  
28860
```

```
prometheus_http_request_duration_seconds_bucket{handler="/",le="+Inf"}  
28860
```

```
prometheus_http_request_duration_seconds_sum{handler="/"}  
1863.80491025699
```

```
prometheus_http_request_duration_seconds_count{handler="/"} 28860
```

Histogram

La parte interesante es el `_bucket`
le es `less` or `equal`

En nuestro ejemplo anterior 26688 tardaron menos o igual a 200ms
`prometheus_http_request_duration_seconds_bucket{handler="/",le="0.2"}`
26688

27760 tomaron menos o igual a 400ms
`prometheus_http_request_duration_seconds_bucket{handler="/",le="0.4"}`
27760

Y hubieron 28860 peticiones en total
`prometheus_http_request_duration_seconds_count{handler="/"}` 28860

Histogram

- > se usan para mostrar cuantiles.
- > Nos dicen que una cierta porcion de eventos tiene un tamaño por debajo de un valor dado.



Histogram

> Por ejemplo el cuantil 0.95 con valor de 300 ms nos dice que el 95% de las peticiones toman menos de 300 ms.



Histogram

> La instrumentación para histogramas es la misma que para summaries.



Histogram

> los buckets se usan para cubrir rangos de 1 ms a 10 s.



Hands on Code



```
$ cat << EOF > Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.network "forwarded_port", guest: 9090, host: 9090
  config.vm.network "forwarded_port", guest: 8000, host: 8000
  config.vm.network "forwarded_port", guest: 8001, host: 8001
  config.vm.network "private_network", ip: "10.5.5.10"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024" # 512
  end
end
EOF
$ vagrant up
$ vagrant ssh
```




```
$ sudo apt-get install python3-pip
```

```
$ sudo nano /etc/environment
```

Agregamos

```
LC_ALL="en_US.UTF-8"
```

```
LC_CTYPE="en_US.UTF-8"
```

```
$ pip3 install prometheus_client
```

```
$ mkdir ~/app && cd ~/app
```



```
$ cat << EOF > server.py
```

```
import http.server
```

```
from prometheus_client import start_http_server
```

```
class MyHandler(http.server.BaseHTTPRequestHandler):
```

```
    def do_GET(self):
```

```
        self.send_response(200)
```

```
        self.end_headers()
```

```
        self.wfile.write(b"Hello World")
```

```
if __name__ == "__main__":
```

```
    start_http_server(8000)
```

```
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
```

```
    server.serve_forever()
```

```
EOF
```

```
$ nano ~/prometheus/prometheus.yml
global:
  scrape_interval: 10s
  scrape_configs:
    - job_name: app
      static_configs:
        - targets:
          - localhost:8000
```



```
$ cd ~/prometheus
```

```
$ python3 server.py
```



python_info



```
$ cat << EOF > server.py
```

```
import http.server
from prometheus_client import start_http_server
from prometheus_client import Counter

REQUESTS = Counter('hello_worlds_total', 'Hello Worlds requested.')

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        REQUESTS.inc()
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")

if __name__ == "__main__":
    start_http_server(8000)
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
    server.serve_forever()
```

```
EOF
```

```
rate(hello_worlds_total[1m])
```

Ejercicio

Crea un script que lance 3 peticiones curl por segundo a localhost:3000


```
import http.server
import random
from prometheus_client import start_http_server
from prometheus_client import Counter

REQUESTS = Counter('hello_worlds_total', 'Hello Worlds requested.')
EXCEPTIONS = Counter('hello_world_exceptions_total', 'Exceptions serving
Hello World.')

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        REQUESTS.inc()
        with EXCEPTIONS.count_exceptions():
            if random.random() < 0.2:
                raise Exception
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")

if __name__ == "__main__":
    start_http_server(8000)
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
    server.serve_forever()
```

```
rate(hello_world_exceptions_total[1m])  
rate(hello_world_exceptions_total[1m])/  
rate(hello_worlds_total[1m])
```

```
import http.server
from prometheus_client import start_http_server
from prometheus_client import Counter

REQUESTS = Counter('hello_worlds_total', 'Hello Worlds requested.')
SALES = Counter('hello_world_sales_total', 'Sales made serving hello
world.')

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        REQUESTS.inc()
        euros = random.random()
        SALES.inc(euros)
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")

if __name__ == "__main__":
    start_http_server(8000)
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
    server.serve_forever()
```

```
rate(hello_world_sales_total[1m])
```

GAUGE



```
import time
from prometheus_client import Gauge

INPROGRESS = Gauge('hello_worlds_inprogress', 'number of HelloWorlds in
progress')
LAST = Gauge('hello_workd_last_time_seconds', 'The last time a Hello
World was served')

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        INPROGRESS.inc()
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")
        LAST.set(time.time())
        INPROGRESS.dec()

if __name__ == "__main__":
    start_http_server(8000)
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
    server.serve_forever()
```

```
hello_world_last_time_seconds  
time() - hello_world_last_time_seconds
```

THE SUMMARY




```
from prometheus_client import Summary

LATENCY = Summary('hello_world_latency_seconds', 'Time for a request  
Hello World.')

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        start = time.time()
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")
        LATENCY.observe(time.time() - start)

if __name__ == "__main__":
    start_http_server(8000)
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
    server.serve_forever()
```

```
hello_world_latency_seconds
hello_world_latency_seconds_count
hello_world_latency_seconds_sum
rate(hello_world_latency_seconds_count[1m])
rate(hello_world_latency_seconds_sum[1m])

rate(hello_world_latency_seconds_count[1m])/rate(hello_world_latency_seconds_sum[1m])
```

THE HISTOGRAM



```
from prometheus_client import Histogram

LATENCY = Histogram('hello_world_latency_seconds', 'Time for a request  
Hello World.')

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        start = time.time()
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")
        LATENCY.observe(time.time() - start)

if __name__ == "__main__":
    start_http_server(8000)
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
    server.serve_forever()
```

```
histogram_quantile(0.95,  
rate(hello_world_latency_seconds_bucket[1m]))
```

LABELS



```
from prometheus_client import start_http_server, Counter

REQUESTS = Counter('hello_worlds_total', 'Hello Worlds requests',
labelnames=['path'])

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        REQUESTS.labels(self.path).inc()
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")

if __name__ == "__main__":
    start_http_server(8000)
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
    server.serve_forever()
```

```
$ curl localhost:8001/login  
$ curl localhost:8001/logout  
$ curl localhost:8001/adduser  
$ curl localhost:8001/comment  
$ curl localhost:8001/view
```

```
http_requests_total{path="/login"}  
http_requests_total{path="/comment"}
```


MULTIPLE LABELS



```
http_requests_total{path="/login"}  
http_requests_total{path="/comment"}
```

Ejercicio

Crea un script que haga tres peticiones aleatorias cada segundo a localhost:8001 a las siguientes paths: login, logout, adduser, comment, view

Con los métodos: POST, GET y DELETE.

```
from prometheus_client import start_http_server, Counter

REQUESTS = Counter('hello_worlds_total', 'Hello Worlds requests',
labelnames=['path', 'method'])

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        REQUESTS.labels(self.path, self.command).inc()
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")

if __name__ == "__main__":
    start_http_server(8000)
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
    server.serve_forever()
```

AGGREGATING



```
rate(hello_worlds_total[5m]  
sum without(path)(rate(hello_worlds_total[5m]))  
sum without(path, instance)(rate(hello_worlds_total[5m]))
```

