Instrumenting Python Applications with Prometheus

Emily Woods & Jessica Greene

Emily Woods

@sometimes_milo

- Site Reliability Engineer
- Previously a chemical engineer
- I <3 Python because of its community
- Sewing, plants and books

Jessica Greene @sleepypioneer

- ★ Software Engine
  - Self Taught/ Community Taught
- ★ Career Changer
  - previously a coffee roaster
  - & Camera assistant
- ★ Fell in love with Python’s versatility
- ★ Community Organiser, Climate activist, knitter, book lover and plant mom

Agenda:

- What is monitoring, why is it important
- Walk through the project we will work on today
- Exposing metrics with prometheus client
  Challenge 1: expose base app metrics on /metrics endpoint
- Adding custom metrics, what makes a meaningful metric?
  Challenge 2: add a custom metric with labels

BREAK

- Inspecting metrics with Prometheus & Grafana
  Challenge 3: Query your metric with PromQL
  Challenge 4: Create a Grafana Dashboard

BONUS

- Data types in prometheus, what do we want to monitor?
  Take home challenge: make a histogram for request latencies

Monitoring: Why?

Image credit: https://getnave.com/blog/kanban-metrics/

Monitoring: What?

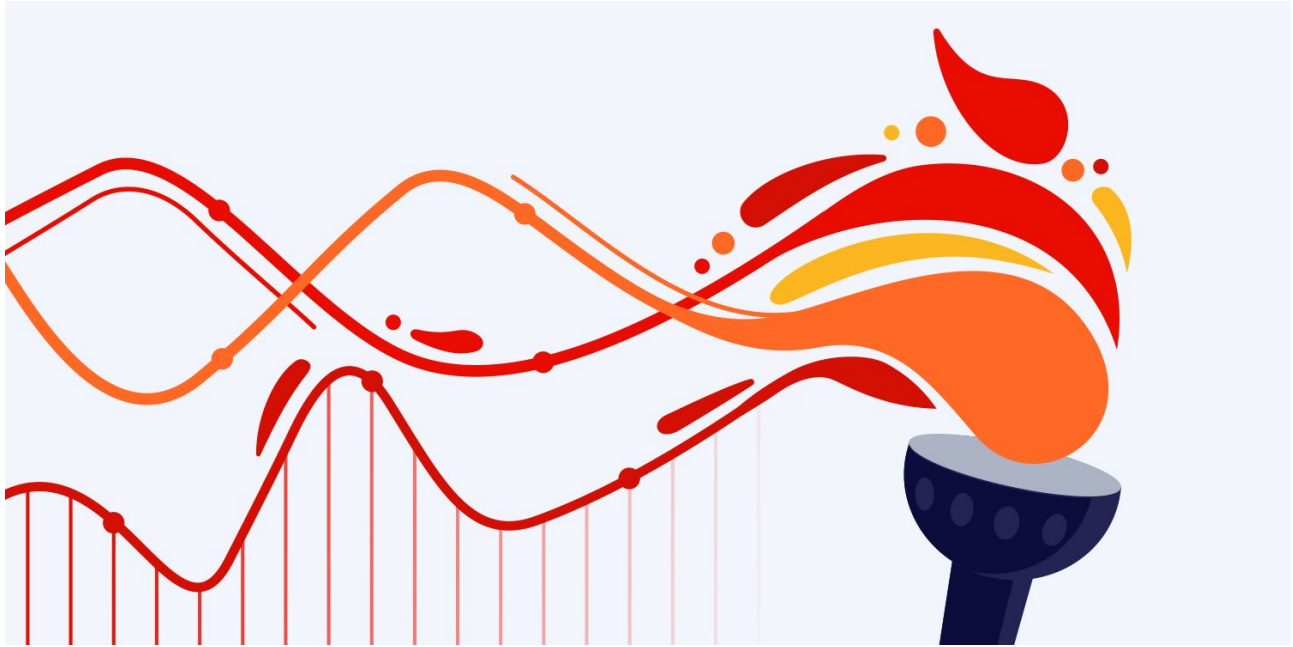
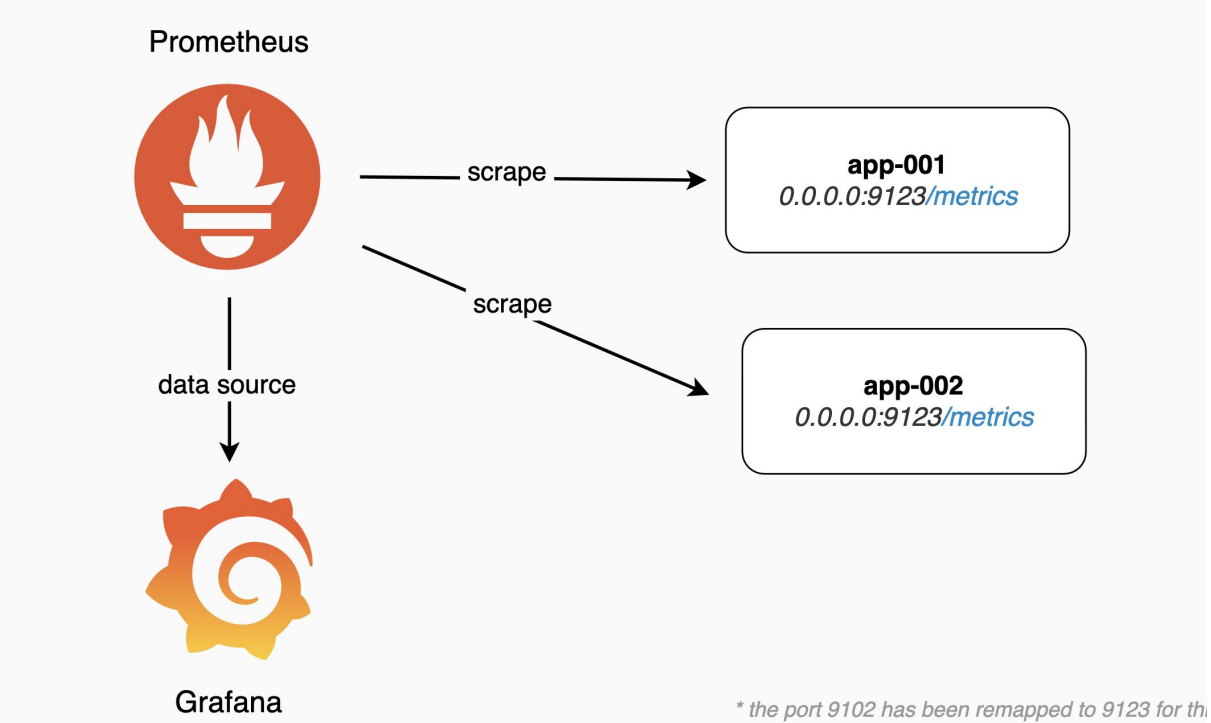


Image credit: https://www.haproxy.com/blog/haproxy-exposes-a-prometheus-metrics-endpoint/

Monitoring: How?



Instrumenting your Python Application

https://github.com/ecosia/python-prometheus-workshop

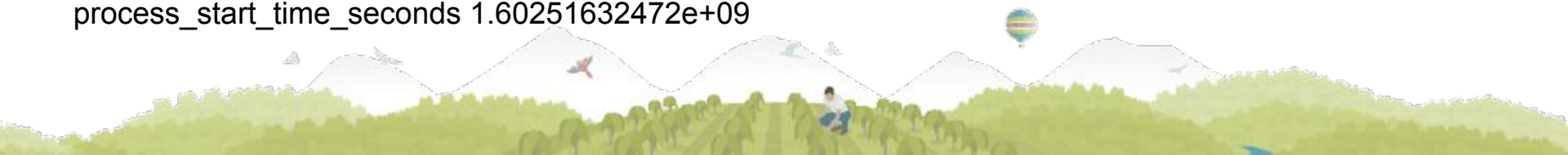
What is a metric? (base metrics)

```
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 1.01
```

```
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 53.0
python_gc_collections_total{generation="1"} 4.0
python_gc_collections_total{generation="2"} 0.0
```

The Python garbage collector has three generations in total, and an object moves into an older generation whenever it survives a garbage collection process on its current generation.

```
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.60251632472e+09
```



https://github.com/ecosia/python-prometheus-workshop/

Challenge 1 - expose metrics

1. Import the python client & use the prometheus handler as a base for your request handler
2. run the application a few times and check /metrics

Defining Custom Metrics

- Useful when ‘out of the box’ metrics aren’t sufficient
- To define one, choose a data type and provide:
  - Base name
  - Description
  - Labels

```
7  from prometheus_client import Counter, MetricsHandler
8
9  c = Counter('requests_total', 'requests', ['status', 'endpoint'])
10
11
12 HOST_NAME = '0.0.0.0' # This will map to avialable port in docker
13 PORT_NUMBER = 8001
14 trees_api_url = "https://api.ecosia.org/v1/trees/count"
15 with open('./templates/treeCounter.html', 'r') as f:
16     html_string = f.read()
17 html_template = Template(html_string)
18
19 def fetch_tree_count():
20     r = requests.get(trees_api_url)
21     # Here is one possible place you may decide to call this metric from
22     c.labels(status=f'{r.status.code}', endpoint='/trees').inc()
23     if r.status_code == 200:
24             return r.json()['count']
25     return 0
```

Defining Custom Metrics

- Useful when ‘out of the box’ metrics aren’t sufficient
- To define one, choose a data type and provide:
  - Base name
  - Description
  - Labels

```
from prometheus_client import Counter, MetricsHandler

c = Counter('requests_total', 'requests', ['status', 'endpoint'])


HOST_NAME = '0.0.0.0' # This will map to avialable port in docker
PORT_NUMBER = 8001
trees_api_url = "https://api.ecosia.org/v1/trees/count"
with open('./templates/treeCounter.html', 'r') as f:
    html_string = f.read()
html_template = Template(html_string)

def fetch_tree_count():
    r = requests.get(trees_api_url)
    # Here is one possible place you may decide to call this metric from
    c.labels(status=f'{r.status.code}', endpoint='/trees').inc()
    if r.status_code == 200:
            return r.json()['count']
    return 0
```

Defining Custom Metrics

- Useful when ‘out of the box’ metrics aren’t sufficient
- To define one, choose a data type and provide:
  - Base name
  - Description
  - Labels

```
7  from prometheus_client import Counter, MetricsHandler
8
9  c = Counter('requests_total', 'requests', ['status', 'endpoint'])
10
11
12 HOST_NAME = '0.0.0.0' # This will map to avialable port in docker
13 PORT_NUMBER = 8001
14 trees_api_url = "https://api.ecosia.org/v1/trees/count"
15 with open('./templates/treeCounter.html', 'r') as f:
16     html_string = f.read()
17 html_template = Template(html_string)
18
19 def fetch_tree_count():
20     r = requests.get(trees_api_url)
21     # Here is one possible place you may decide to call this metric from
22     c.labels(status=f'{r.status.code}', endpoint='/trees').inc()
23     if r.status_code == 200:
24             return r.json()['count']
25     return 0
```

Defining Custom Metrics

- Useful when ‘out of the box’ metrics aren’t sufficient
- To define one, choose a data type and provide:
  - Base name
  - Description
  - Labels

```python
from prometheus_client import Counter, MetricsHandler

c = Counter('requests_total', 'requests', ['status', 'endpoint'])


HOST_NAME = '0.0.0.0' # This will map to avialable port in docker
PORT_NUMBER = 8001
trees_api_url = "https://api.ecosia.org/v1/trees/count"
with open('./templates/treeCounter.html', 'r') as f:
    html_string = f.read()
html_template = Template(html_string)

def fetch_tree_count():
    r = requests.get(trees_api_url)
    # Here is one possible place you may decide to call this metric from
    c.labels(status=f'{r.status.code}', endpoint='/trees').inc()
    if r.status_code == 200:
            return r.json()['count']
    return 0
```

Defining Custom Metrics

- Useful when ‘out of the box’ metrics aren’t sufficient
- To define one, choose a data type and provide:
  - Base name
  - Description
  - Labels

```python
from prometheus_client import Counter, MetricsHandler

c = Counter('requests_total', 'requests', ['status', 'endpoint'])


HOST_NAME = '0.0.0.0' # This will map to avialable port in docker
PORT_NUMBER = 8001
trees_api_url = "https://api.ecosia.org/v1/trees/count"
with open('./templates/treeCounter.html', 'r') as f:
    html_string = f.read()
html_template = Template(html_string)

def fetch_tree_count():
    r = requests.get(trees_api_url)
    # Here is one possible place you may decide to call this metric from
    c.labels(status=f'{r.status.code}', endpoint='/trees').inc()
    if r.status_code == 200:
            return r.json()['count']
    return 0
```

Querying Custom Metrics

```
# HELP requests_total Total requests
# TYPE requests_total counter
requests_total{method="get",status="200"} 1.0
```

Description

Querying Custom Metrics

```
# HELP requests_total Total requests
# TYPE requests_total counter
requests_total{method="get",status="200"} 1.0
```

Measurement type

Querying Custom Metrics

```
# HELP requests_total Total requests
# TYPE requests_total counter
requests_total{method="get",status="200"} 1.0
```

Base name

Custom Metrics

```
# HELP requests_total Total requests
# TYPE requests_total counter
requests_total{method="get",status="200"} 1.0
```

Labels

https://github.com/ecosia/python-prometheus-workshop/

Challenge 2 - custom metric

1. add request counter with status code as label & look at /metrics
2. run the application a few times and check again /metrics
3. Add a label to your metric for status code

BREAK

Scraping metrics & creating dashboards

App:

http://localhost:8001

Prometheus:

http://localhost:9090

Grafana:

http://localhost:3000

Monitoring your metrics

https://github.com/ecosia/python-prometheus-workshop

docker-compose up

Status > Targets

Prometheus Alerts Graph Status Help

Targets

All Unhealthy

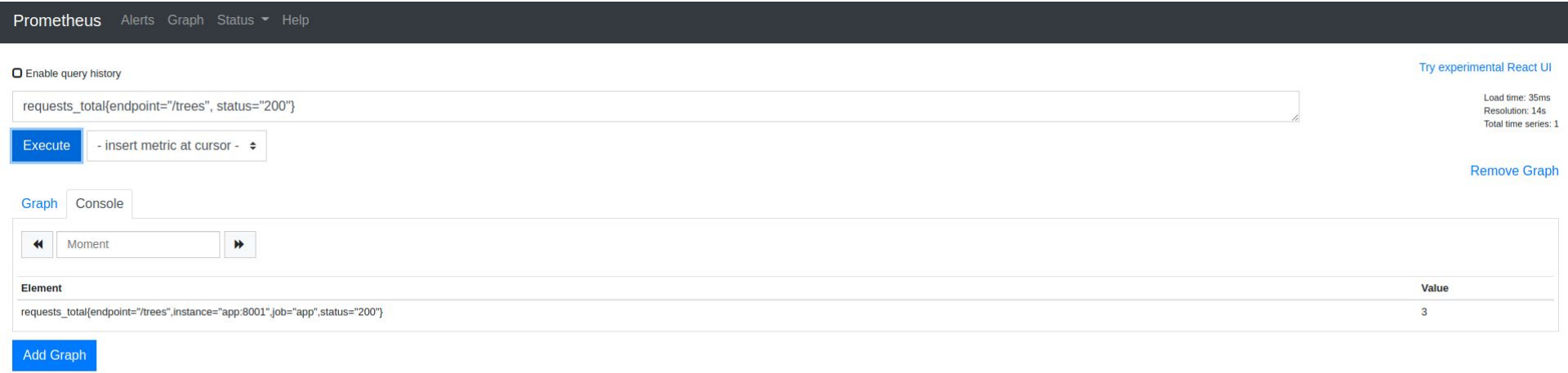
app (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://app:8001/metrics	UP	instance="app:8001" job="app"	6.785s ago	1.81ms	

prometheus (1/1 up) show less

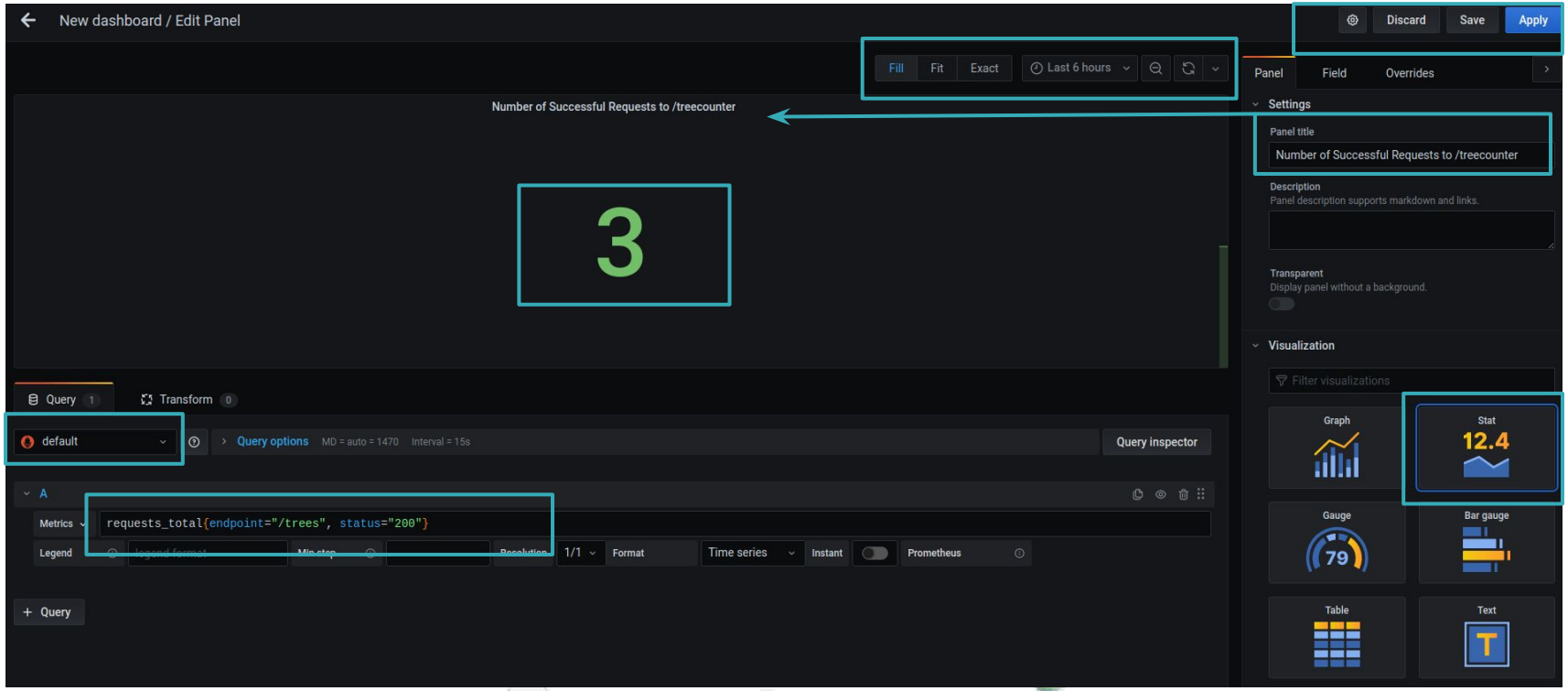
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	922ms ago	15.97ms	

Using PromQL to query Metrics



requests_total{endpoint="/trees", status="200"}

Creating a panel in your dashboard



https://github.com/ecosia/python-prometheus-workshop/

Challenge 3 - Query your metric with PromQL *(section 3 of readme)*

1. Run the app and prometheus with docker-compose
2. Query your metrics in the prometheus UI using PromQL

https://github.com/ecosia/python-prometheus-workshop/

Challenge 4 - Create a Grafana Dashboard *(section 3 of readme)*

1. Run the app, prometheus and Grafana with docker-compose
2. Go to localhost:3000
3. Create a Grafana Dashboard
4. Create a panel to visualise your customer metric (split by labels, irate or cumulative)

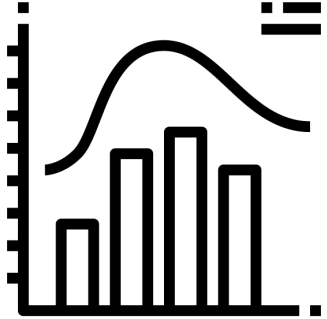
Data types in Prometheus

Created by Vectors Point
from Noun Project

1. Counter

Created by IconMark
from Noun Project

2. Gauge



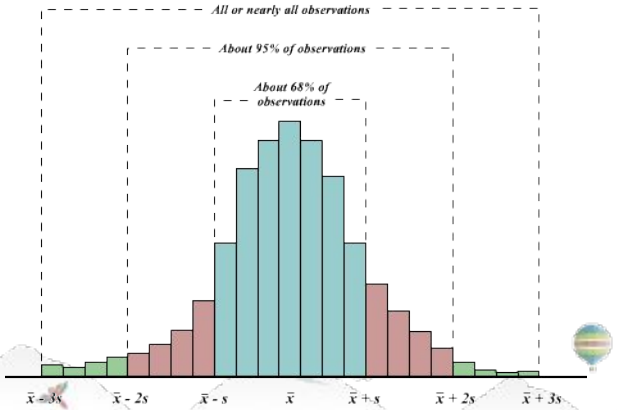
Created by Nhor
from Noun Project

3. Histogram

4. Summary

Histogram

- In Prometheus, a histogram measures the frequency of value observations that fall into `buckets`.
- Useful when you want approximations over a known range of values
- Common uses: measuring response duration, request size



What a Histogram exposes

- **Cumulative counters** for the observation buckets, exposed as `<metric_name>_bucket`

```
request_latency_seconds_bucket{endpoint="/treecounter",le="0.1"} 1.0
request_latency_seconds_bucket{endpoint="/treecounter",le="0.5"} 4.0
request_latency_seconds_bucket{endpoint="/treecounter",le="1.0"} 5.0
request_latency_seconds_bucket{endpoint="/treecounter",le="+Inf"} 5.0
```

- The **total sum** of all observed values, exposed as `<metric_name>_sum`

```
request_latency_seconds_sum{endpoint="/treecounter"} 1.14
```

- The **count** of events that have been observed, exposed as `<metric_name>_count` (identical to `<metric_name>_bucket{le="+Inf"}`)

```
request_latency_seconds_count{endpoint="/treecounter"} 5.0
```

https://prometheus.io/docs/concepts/metric_types/

https://github.com/ecosia/python-prometheus-workshop/

Bonus Challenge

1. make a histogram for request latencies.

(hint for measuring latencies with time)

Solution in branch: *solution_challenge_5*

@ecosia
@sometimes_milo
@sleepypioneer

Go forth and monitor!

Gif credit: https://giphy.com/tonybabel

Resources

https://prometheus.io

https://prometheus.io/docs/practices/histograms/

https://grafana.com/

https://tomgregory.com/the-four-types-of-prometheus-metrics/

https://github.com/ecosia/python-prometheus-workshop