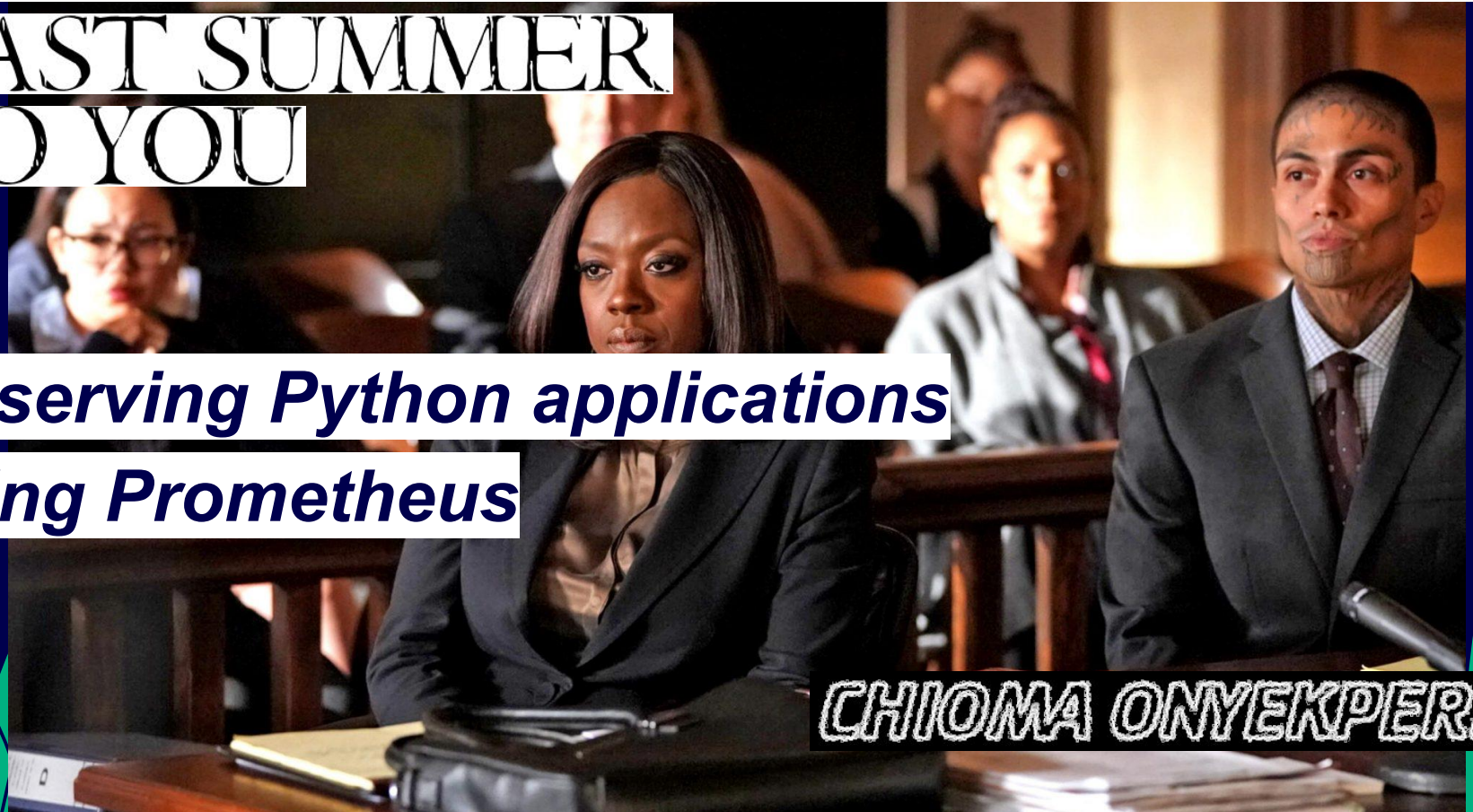
WE KNOW WHAT YOUR APP DID LAST SUMMER DO YOU



Observing Python applications using Prometheus

CHIOMA ONYEKPERE

CHIOMA ONYEKPERE

She/her

- Full-Stack Engineer at Astronomer
- Digital nomad. From fear of “magic” dark screen with coloured text to becoming a magician.
- Tech collaborator. Happy to strengthen the underrepresented community through mentorship

@simpcyclassy

Agenda

1. What is Monitoring & Why is it important?
2. Overview of today's project repo
3. Exposing metrics with Prometheus Client
    a. **Challenge 1**: Expose base app metrics on /metrics endpoint
4. Adding custom metrics: What makes a meaningful metric?
    a. **Challenge 2**: Adding a custom metric with labels
5. Mid point QA
6. Inspecting metrics with Prometheus & Grafana
    a. **Challenge 3**: Query your metric with PromQL
    b. **Challenge 4:** Create a Grafana Dashboard
7. Q&A

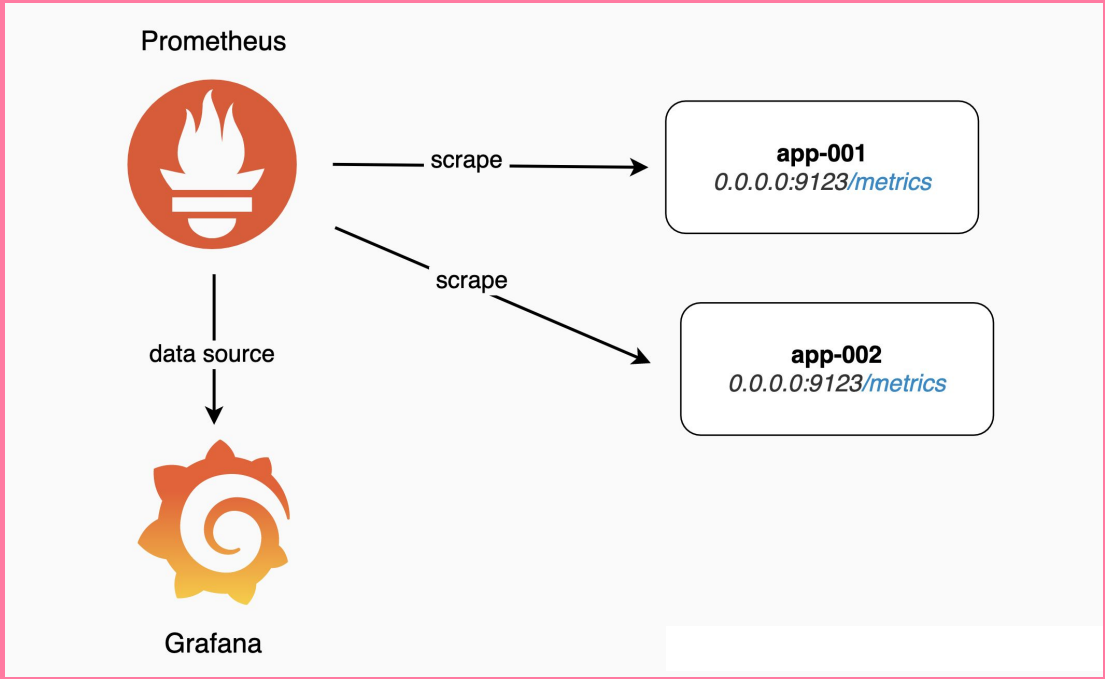
Monitoring: What is it?

@simpcyclassy

Monitoring: Why does it matter?

@simpcyclassy

Monitoring: How?



@simpcyclassy

@simpcyclassy

Tour

https://github.com/Simpcyclassy/PyConES23-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
```

@simpcyclassy

Challenge 1 - expose base metrics

1. Import **MetricsHandler** from the prometheus python client (**prometheus_client**)
2. Remember to remove **BaseHTTPRequestHandler**
3. Use the prometheus **MetricsHandler** as a base for your request handler
4. Once you've added the code, check it's working by running "make dev" and refresh the application a few times and check /metrics

https://github.com/Simpcyclassy/PyConES23-prometheus-workshop

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
```

@simpcyclassy

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
```

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
```

Calling the Custom Metrics

- We use the increment method to call our custom metric counter

** Note there are different places where we can place this, and the placement has an effect on what the metric tells us**

```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
```

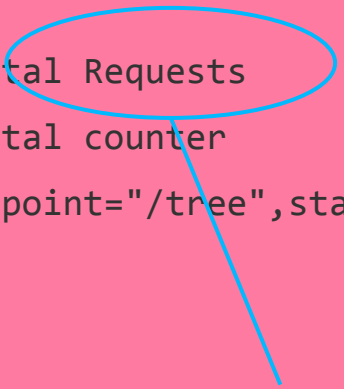
@simpcyclassy

How Custom Metrics Look

```
# HELP requests_total Requests
# TYPE requests_total counter
requests_total{endpoint="/tree",status="200"} 1.0
```

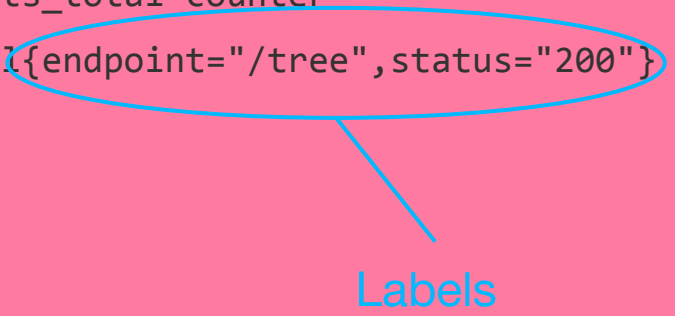
How Custom Metrics Look

```
# HELP requests_total Requests
# TYPE requests_total counter
requests_total{endpoint="/tree",status="200"} 1.0
```



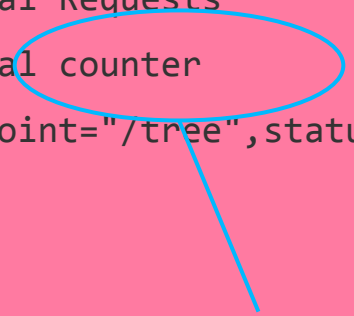
How Custom Metrics Look

```
# HELP requests_total Requests
# TYPE requests_total counter
requests_total{endpoint="/tree",status="200"} 1.0
```



How Custom Metrics Look

```
# HELP requests_total Requests
# TYPE requests_total counter
requests_total{endpoint="/tree",status="200"} 1.0
```



Challenge 2 - custom metrics

1. Add a custom counter metric. It will count requests with the status code as a label.
2. Re-run the server by stopping it and then re-starting it.
3. Refresh the main application page (/treecounter) a few times and check /metrics

https://github.com/Simpcyclassy/PyConES23-prometheus-workshop

Questions so far?

@simpcyclassy


And now for something completely different
Metric Queries &
creating dashboards

Scraping metrics & creating dashboards

@simpcyclassy

Monitoring your metrics

Prometheus:

http://localhost:9090

App:

http://localhost:8001

Grafana:

http://localhost:3000

https://github.com/ecosia/pycon23-prometheus-workshop

docker-compose up

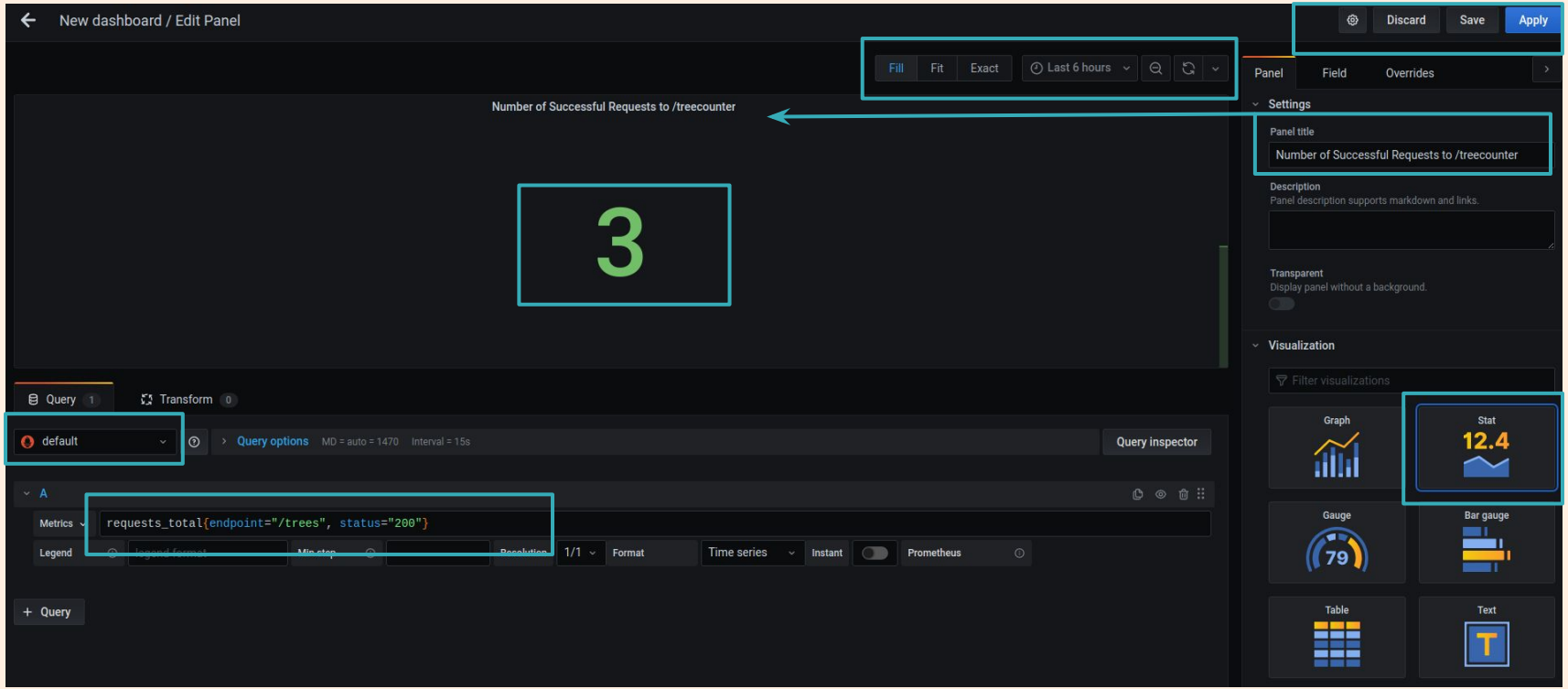
Challenge 3 - PromQL queries

(section 3 of readme)

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

https://github.com/Simpcyclassy/PyConES23-prometheus-workshop

Creating a panel in your dashboard



Challenge 4 - Build a Dashboard

(section 3 of readme)

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

https://github.com/Simpcyclassy/PyConES23-prometheus-workshop

@simpcyclassy

Q&A

@simpcyclassy

Go forth and monitor!

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/pycon23-prometheus-workshop

@simpcyclassy

@simpcyclassy

Thank You

Chioma Onyekpere

Software Engineer at Astronomer