

OpenTelemetry metrics for Python without the remorse i.e. with eBPF

An introduction to Grafana Beyla



Mario Macías @MaciasUPC

Nikola Grcevski

PyCon 2023

Dublin, Ireland

Table of contents

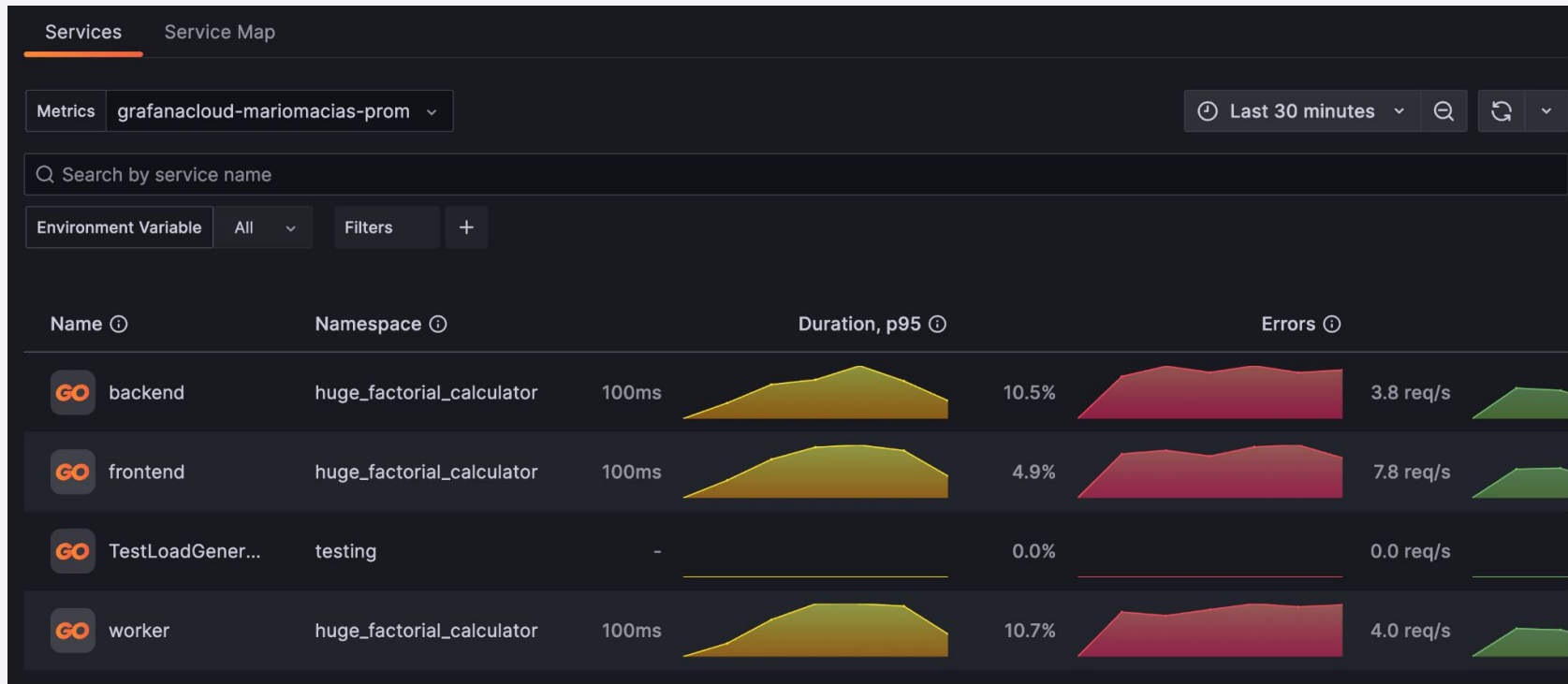
- Introduction
- OpenTelemetry
- eBPF
- Grafana Beyla
- Performance comparison
- Conclusions



Introduction

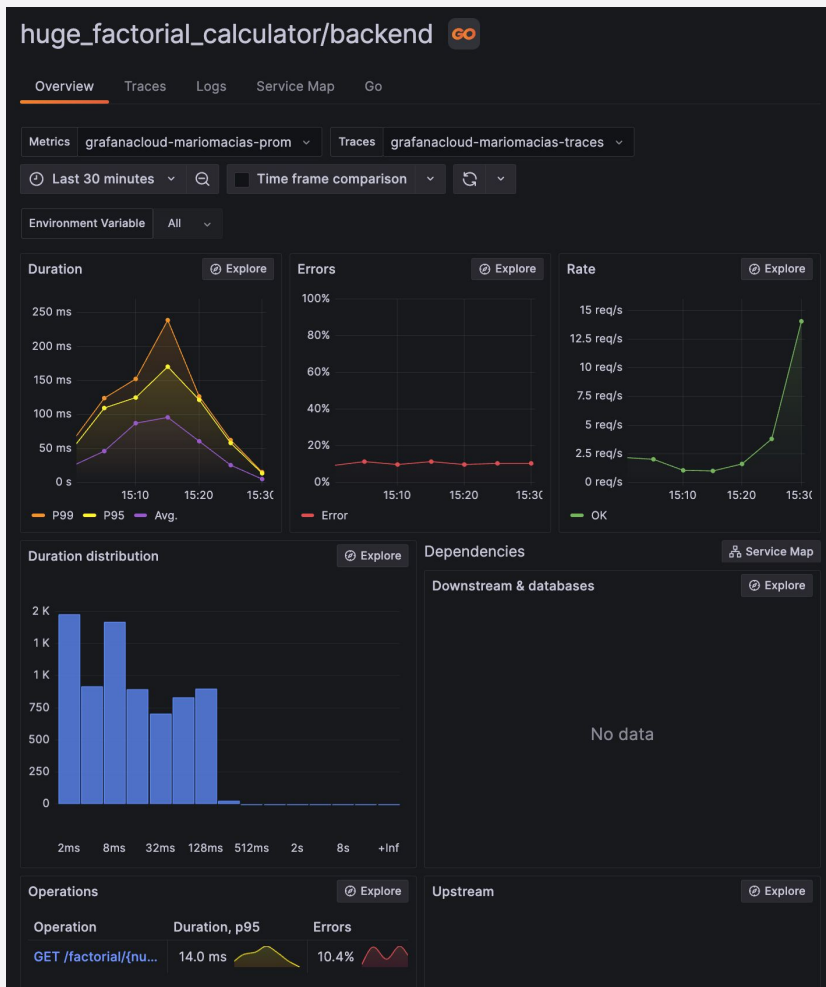
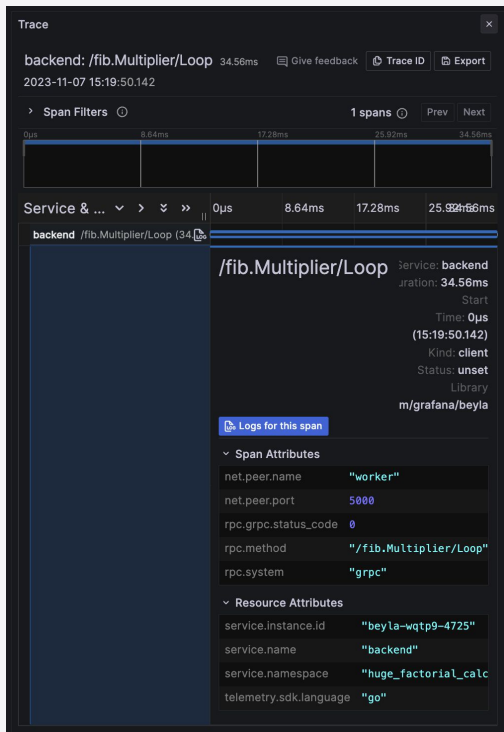
Grafana AppO11y

- App(lication) O(bservabilit)y

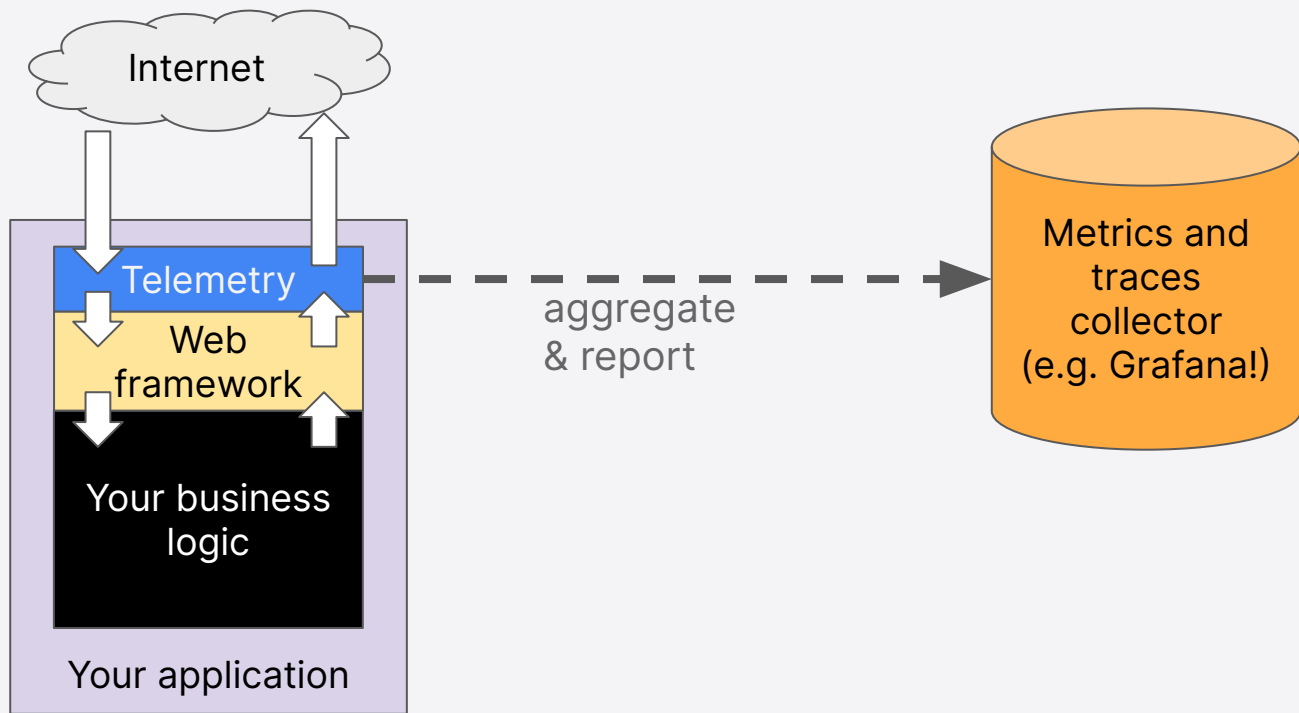


Grafana AppO11y

- App(lication) O(bservability)



Requirement: your application must be instrumented





OpenTelemetry

- Collection of APIs, SDKs, and tools
 - Data format
 - Remote API format
 - Set of language libraries
- Instrument, generate, collect, and export telemetry data
 - Metrics
 - Logs
 - Traces
- Open
- Vendor neutral



OpenTelemetry & Python

- Automatic instrumentation

```
pip install opentelemetry-distro opentelemetry-exporter-otlp  
opentelemetry-bootstrap -a install
```

```
OTEL_SERVICE_NAME=your-service-name \  
OTEL_TRACES_EXPORTER=console,otlp \  
OTEL_METRICS_EXPORTER=console \  
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=0.0.0.0:4317 \  
opentelemetry-instrument python myapp.py
```



OpenTelemetry & Python

- Manual instrumentation

```
provider = TracerProvider()
processor = BatchSpanProcessor(ConsoleSpanExporter())
provider.add_span_processor(processor)
trace.set_tracer_provider(provider)
tracer = trace.get_tracer("my.tracer.name")

def do_work():
    with tracer.start_as_current_span("span-name") as span:
        print("doing some work...")
        # When the 'with' block goes out of scope, 'span' is closed for you
```





eBPF

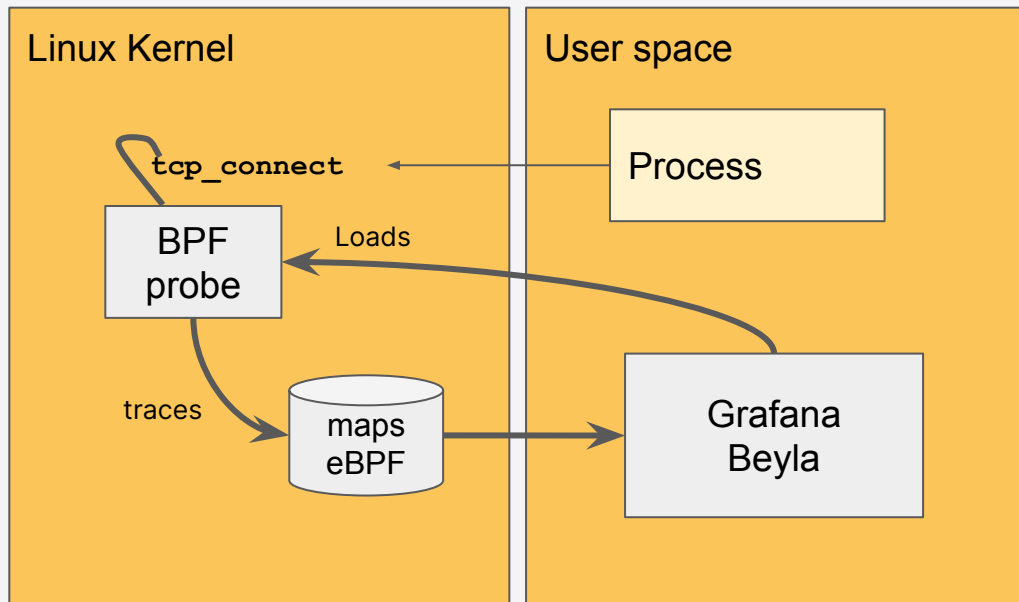
eBPF

- JIT Virtual Machine at the Linux Kernel
 - Make Linux Kernel programmable!
- Can hook your probe programs to multiple events of the Kernel, libraries and user-space programs
 - Lets you see (and even modify) the runtime memory



Example: trace each new TCP connection

```
int tcp_connect(struct sock *sk) ;
```



eBPF advantages

- Fast: JIT compilation
- Stable
 - Programs are pre-verified before loading
 - Prevent unallowed memory accesses
 - Prevent memory loops
- Clean
 - Stopping (or crashing) the “monitor” frees the loaded resources.



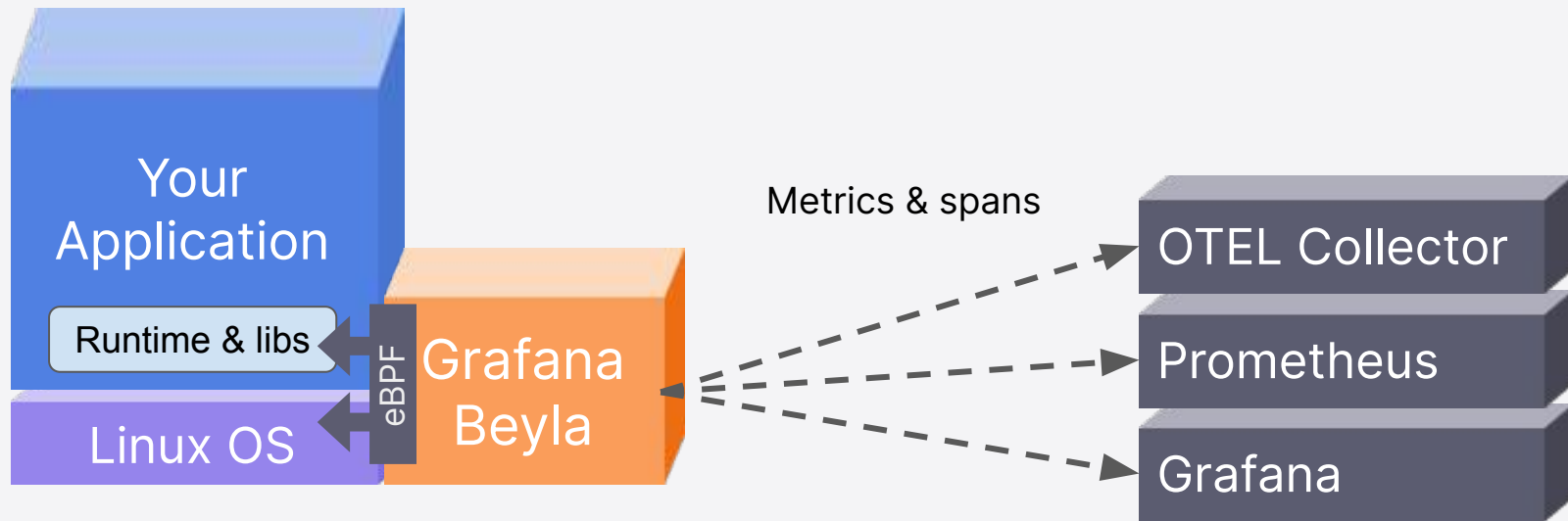
eBPF Disadvantages

- Hard to debug (kprintf)
- Your probes are often dependent of the implementation details
 - Arguments in stack vs registers
 - Architecture/language/compiler conventions
 - Big endian vs little endian
 - etc...
- Changes in the inspected APIs can break your code
- User-space monitor program requires at least CAP_SYS_ADMIN privileges





Beyla native eBPF auto-instrumentation



Running Beyla

```
export BEYLA_EXECUTABLE_NAME=.
```

```
export OTEL_EXPORTER_OTLP_ENDPOINT=0.0.0.0:4317
```

```
sudo -E beyla
```



Performance considerations

Motivation: is Python OpenTelemetry slow?

<https://github.com/GoogleCloudPlatform/bank-of-anthos/issues/356>

- They didn't use batch processor
- Simple Export Span processor:
One connection/export per trace!

All Python services have very slow HTTP requests #356



djmailhot opened this issue on Sep 19, 2020 · 3 comments · Fixed by #360



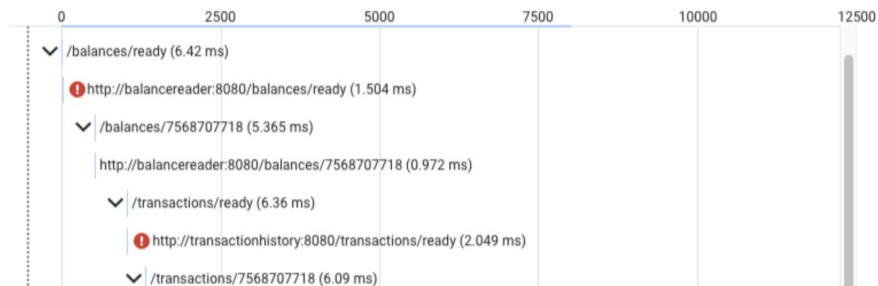
djmailhot commented on Sep 19, 2020

Contributor ...

After fiddling with this for a long while, I'm inclined to believe **this unexpected latency is common to all our Python services.**

I inserted HTTP calls to the /ready endpoint for the contactservice and the userservice, and both calls have large latencies even though the function is simply `return "ok", 200`.

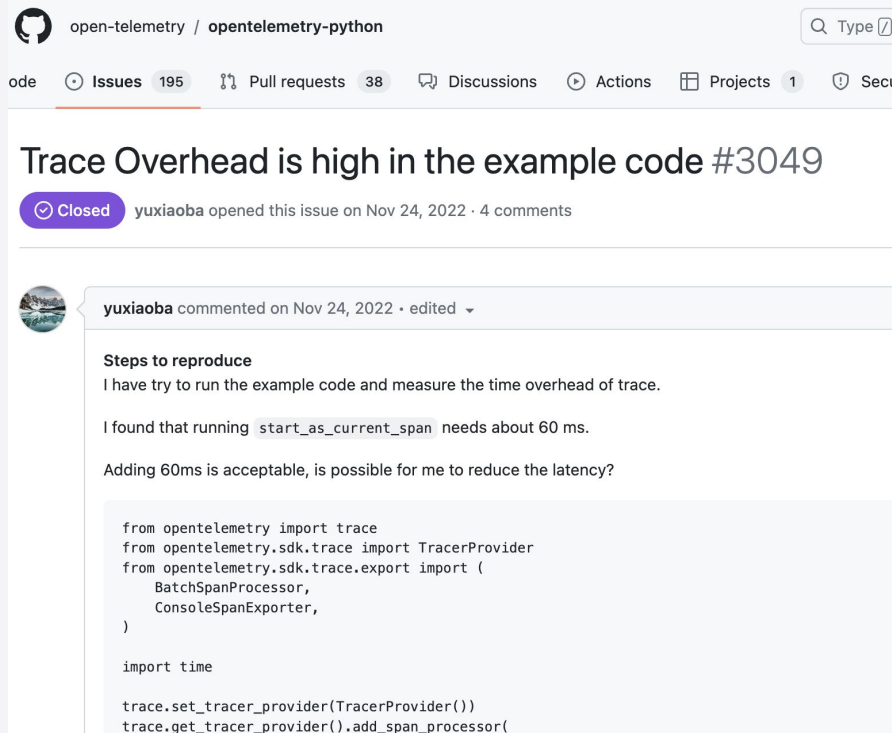
☐ Show Logs ⓘ No Logs found for this trace



Is Python OpenTelemetry slow? (II)

<https://github.com/open-telemetry/opentelemetry-python/issues/3049>

- Problem was partially in an error in the Benchmark creation
- But there is still visible overhead of Running OpenTelemetry
- Suggestion: add sampling



The screenshot shows a GitHub issue page for the repository 'open-telemetry / opentelemetry-python'. The issue title is 'Trace Overhead is high in the example code #3049', marked as 'Closed'. It was opened by user 'yuxiaoba' on Nov 24, 2022, with 4 comments. A comment from 'yuxiaoba' dated Nov 24, 2022, includes the following text:

Steps to reproduce
I have try to run the example code and measure the time overhead of trace.

I found that running `start_as_current_span` needs about 60 ms.

Adding 60ms is acceptable, is possible for me to reduce the latency?

```
from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import (
    BatchSpanProcessor,
    ConsoleSpanExporter,
)

import time

trace.set_tracer_provider(TracerProvider())
trace.get_tracer_provider().add_span_processor(
```



Is Python OpenTelemetry slow? (III)

Spoiler: **no**, it isn't.

... but can we do it better?

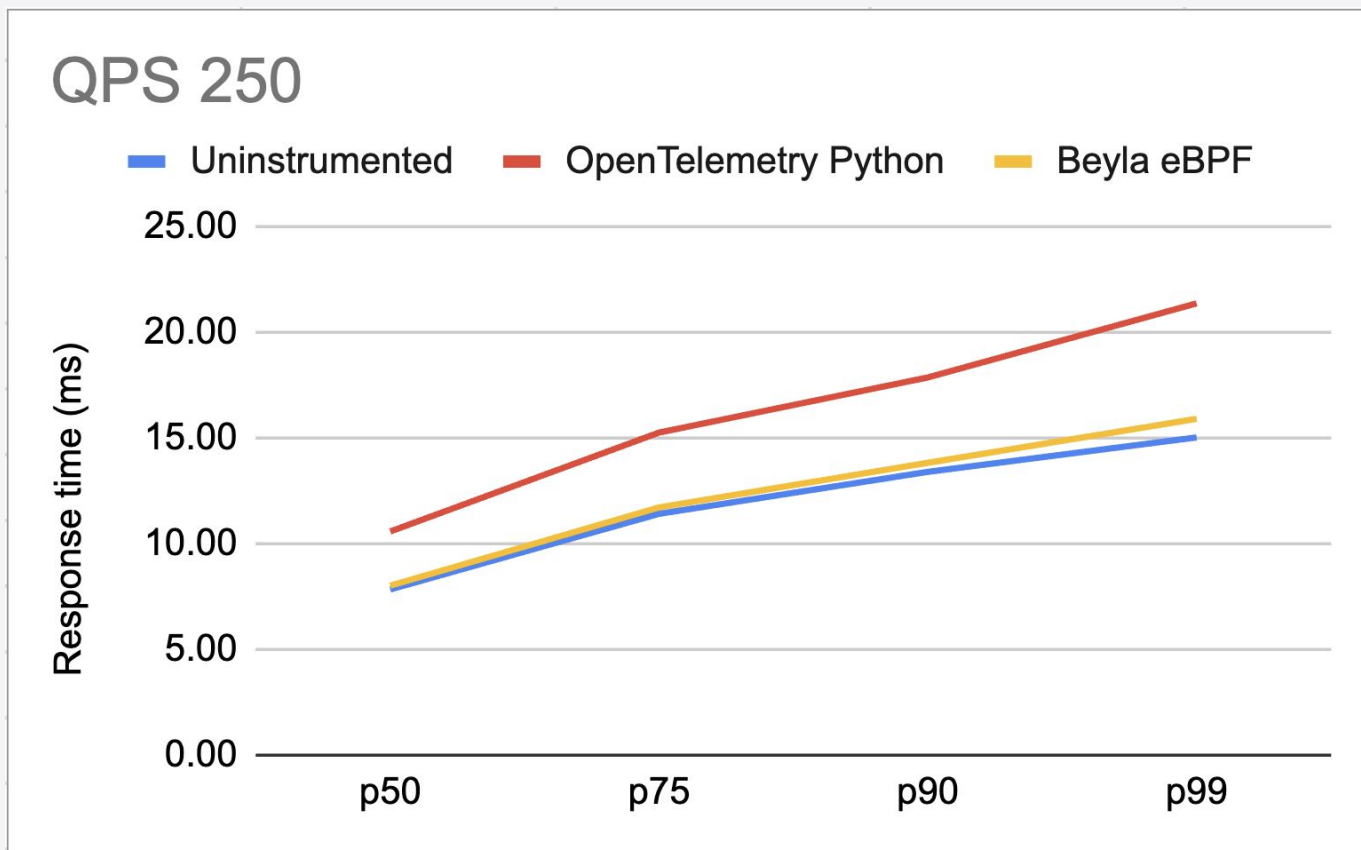


Testing scenario

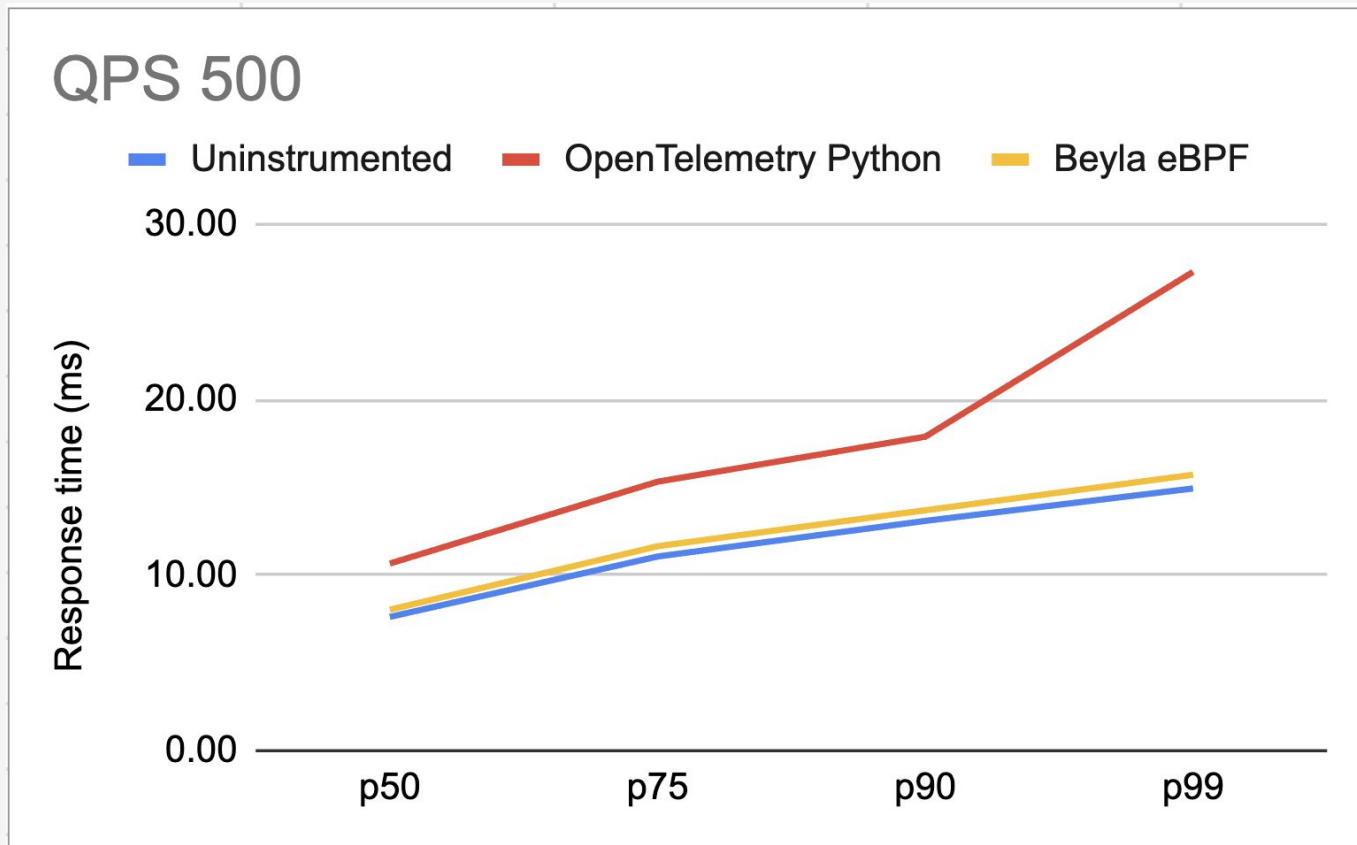
- HTTP server
- We eliminated as much as noise as possible
 - Bare metal hardware, avoiding overhead of virtualization/docker-proxy network latencies
 - Pinning to separate cores to be able to get fair estimates of CPU utilization
 - Turning off CPU throttling and turbo boost
- Compared 3 scenarios
 - Uninstrumented service
 - Service instrumented with OpenTelemetry Flask autoinstrumenter
 - Service instrumented with Beyla



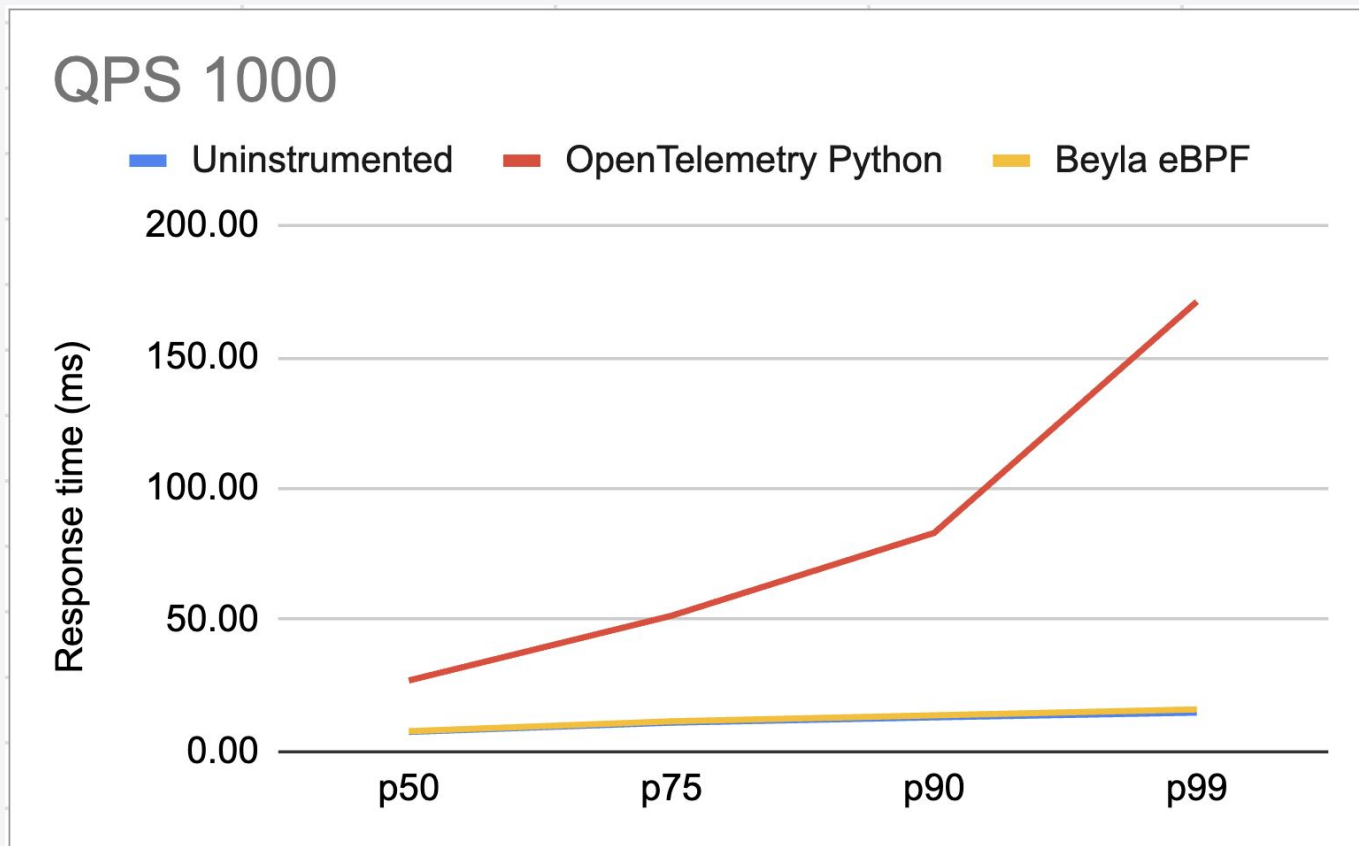
Performance comparison



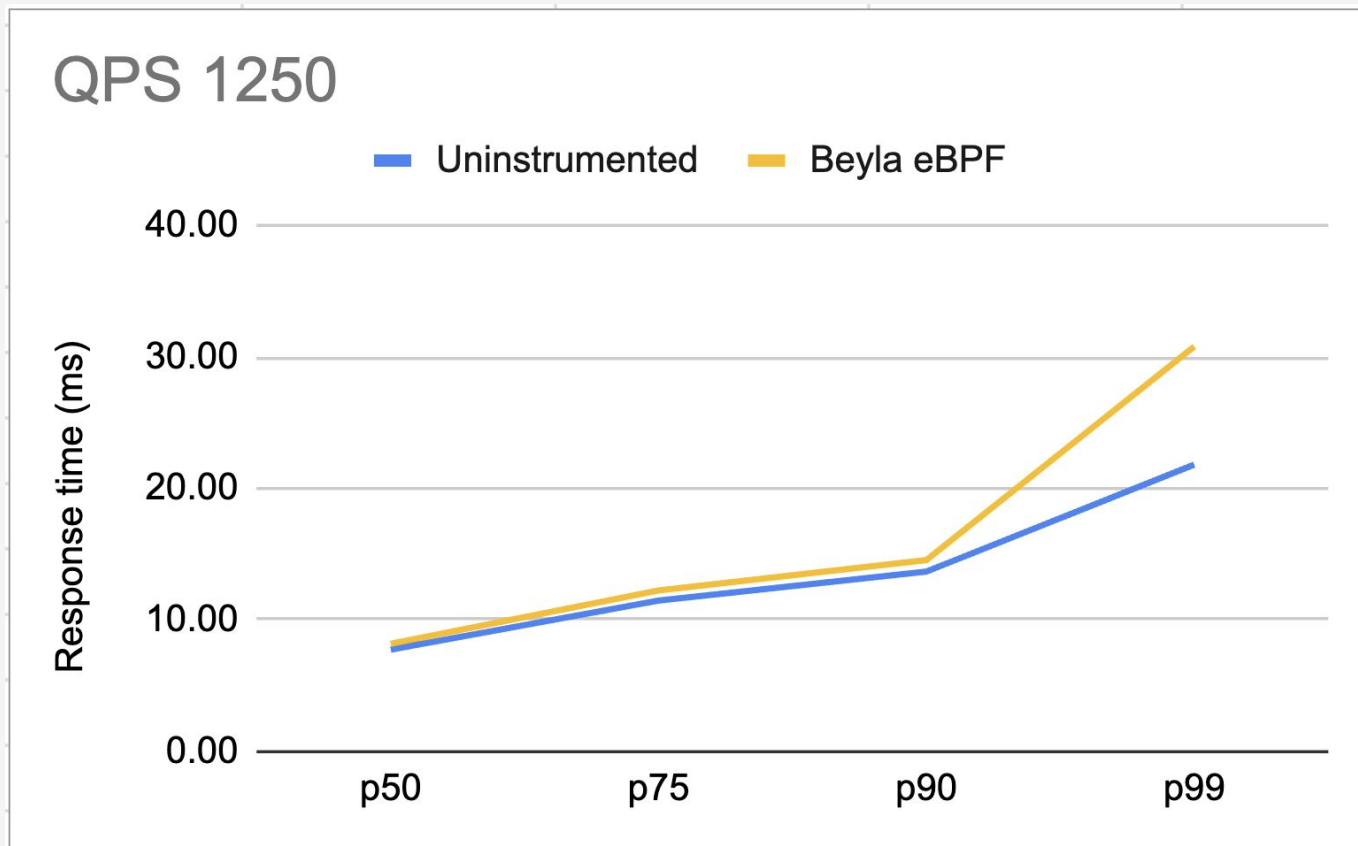
Performance comparison



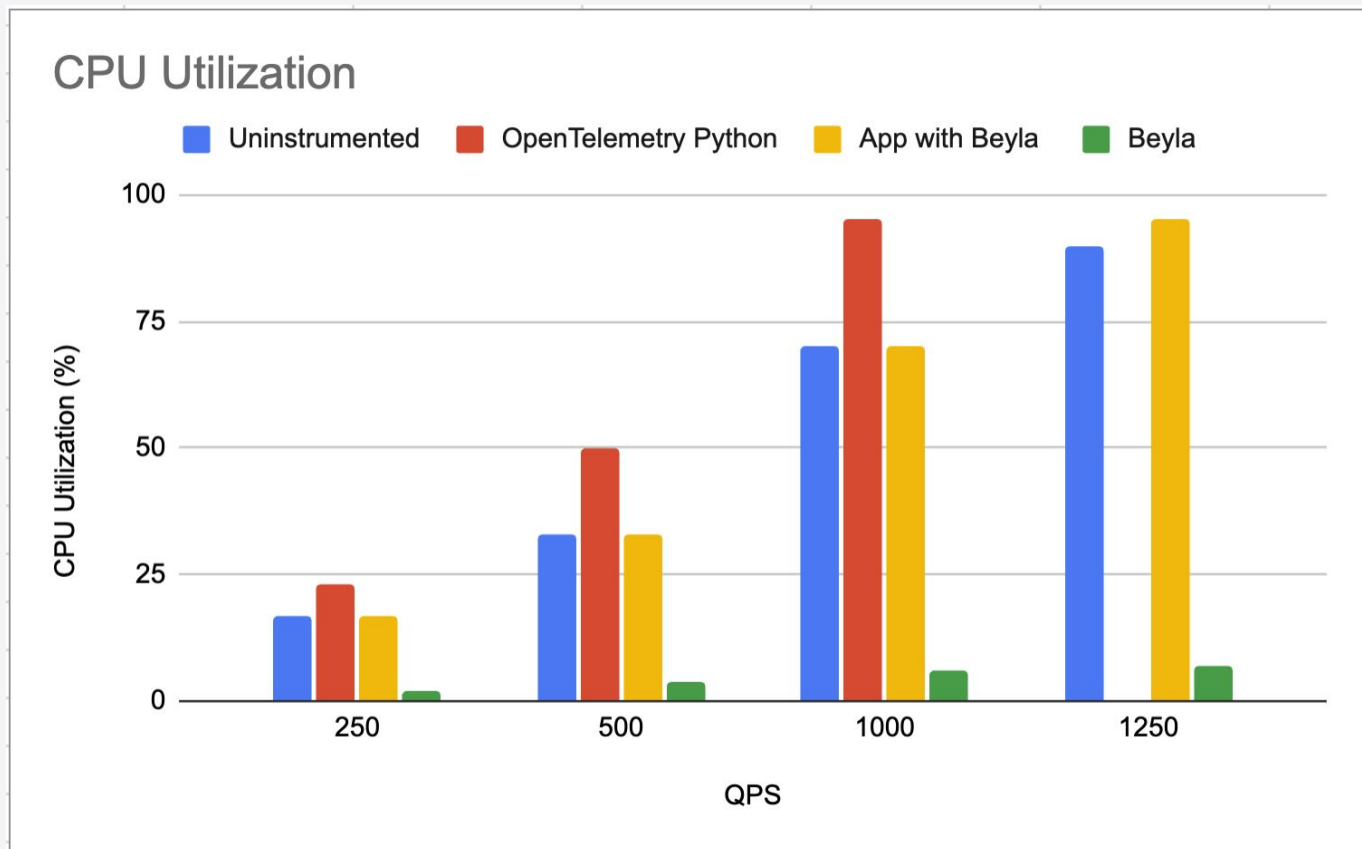
Performance comparison



Performance comparison



Impact in resources



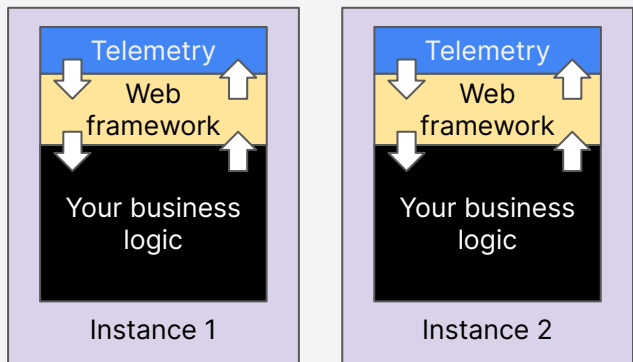
Conclusions

Beyla vs OpenTelemetry SDKs

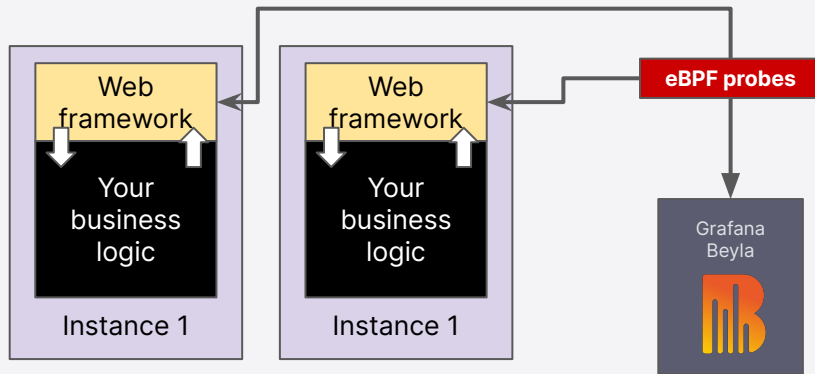
- Use OpenTelemetry SDKs when...
 - You need fine-grained details about your traces
 - Performance is not a blocker
- Use Beyla when...
 - You can't spend time instrumenting your legacy applications
 - Your language/runtime version is unsupported by the SDKs
 - Performance concerns



Decoupling trace/metric generation from your workload



If observability slows down,
you need to scale your app



If observability slows down,
you can scale Beyla



Thank you for your attention!

Beyla 1.0 General Availability
November 15th

<https://grafana.com/oss/beyla-ebpf/>
<https://github.com/grafana/beyla>