Why does continuous profiling matter to developers?

KubeCon NA Salt Lake City 2024
Co-located event: AppDeveloperCon

Who are we?

Jonas Kunz

Observability SW Engineer @ Elastic

OpenTelemetry-Java Contributor

Jonas Kunz @ CNCF-Slack

Mauricio Salatino

@Diagrid @Daprdev

Application Development WG co-chair

https://salaboy.com

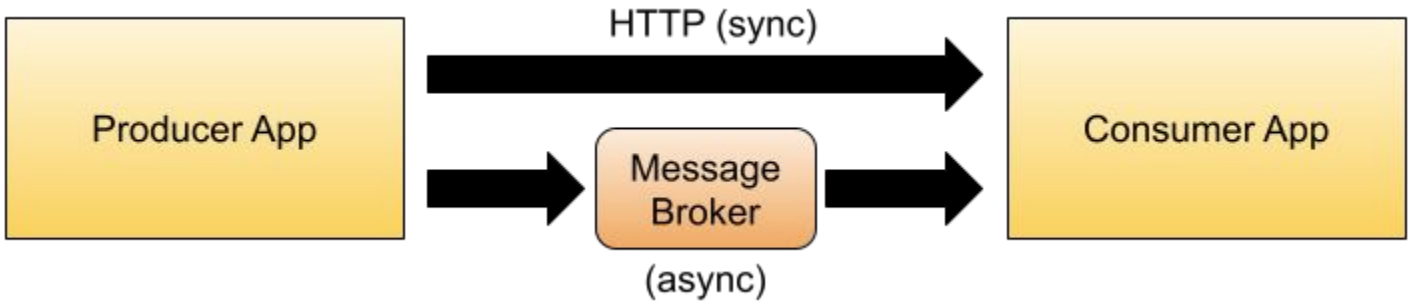
Application Development WG Survey

ADD QR CODE

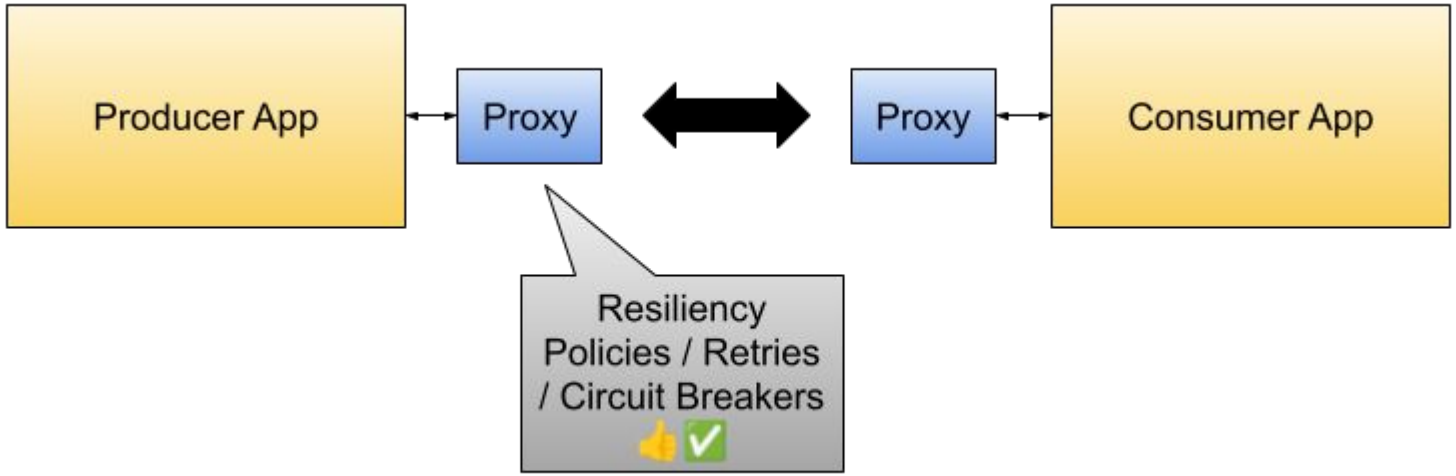
Agenda

- Building Cloud Native Resilient and Observable applications
- Pillars of Observability
- (Continuous) Profiling
- Next steps

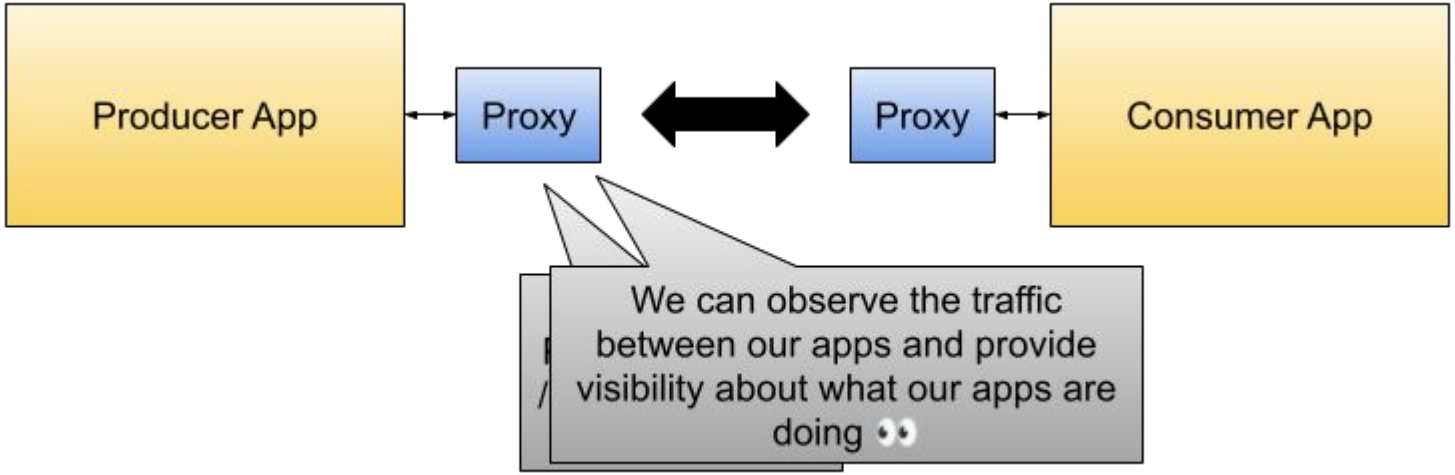
Distributed applications 101



Cloud Native distributed applications



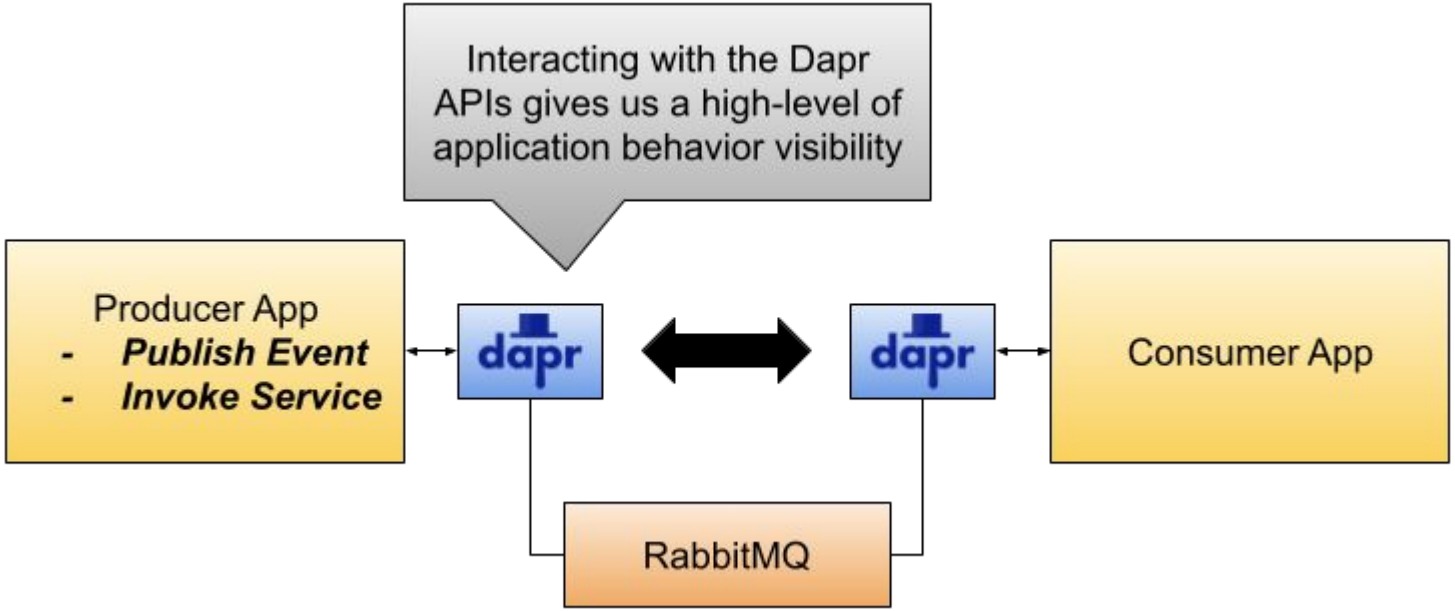
Resilient and observable distributed applications



Dapr: enabling developers with APIs

- Stands for **D**istributed **ap**plication **R**untime
- Uses a proxy to expose application-level APIs to solve common distributed challenges
- We used two APIs for this examples:
  - Service to Service invocation
  - PubSub for async communications
- All APIs cover cross-functional concerns such as resiliency, observability and security

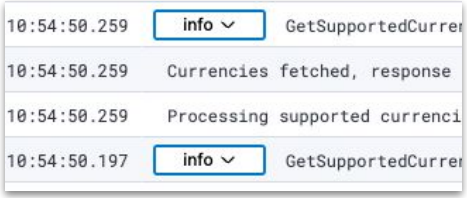
With Dapr


Interacting with the Dapr APIs gives us a high-level of application behavior visibility
Producer App
- Publish Event
- Invoke Service
dapr
dapr
Consumer App
RabbitMQ

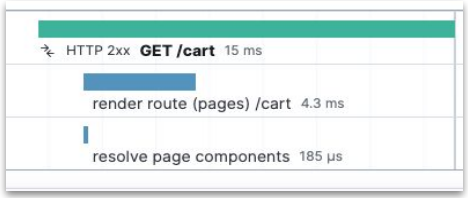
Demo #1

Let’s use our apps!

The 3 Pillars of Observability



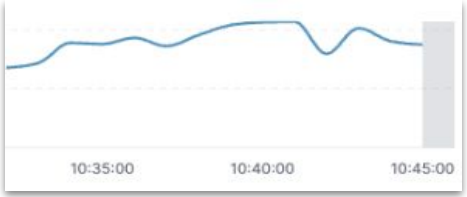
Logs



Traces

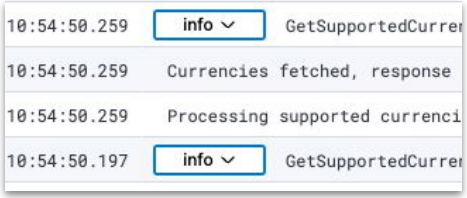
OpenTelemetry

since 2019

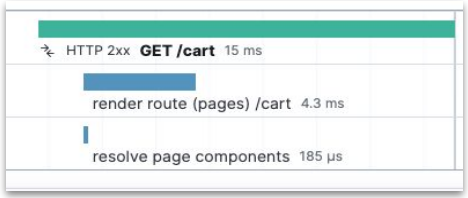


Metrics

The ~~3~~ 4 Pillars of Observability



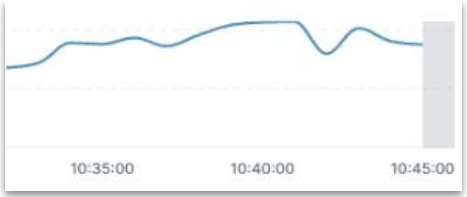
Logs



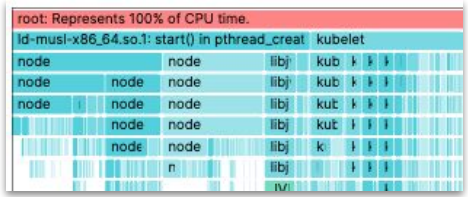
Traces

OpenTelemetry

since 2019



Metrics



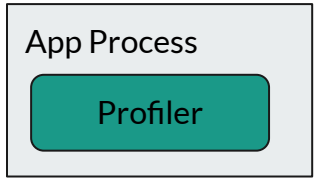
Profiles

since 2022

Profiling

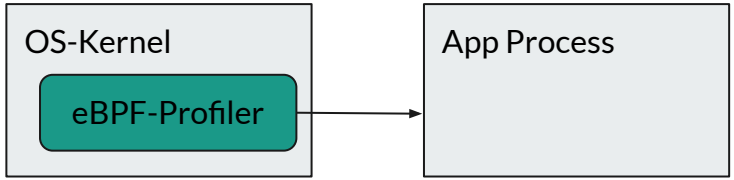
- Measuring where and how an application spends it's time **without** having to modify/instrument it
- "Time" can be many things
  - CPU-Time, Wall-Clock, IO-time, …
- Profiling sees the world from OS-perspective (Threads, processes, OS-resources)

In-Process Profiling



E.g. Linux Perf, Java Flight Recorder

eBPF Profiling



E.g. opentelemetry-ebpf-profiler

Continuous-Profiling in Production

- Optimyze.cloud launches low-overhead multi-runtime zero-instrumentation profiler in 2021
- Acquired by Elastic soon after
- Donated to OpenTelemetry in 2024
- Continued development and evolution

opentelemetry-ebpf-profiler

Whole-System Visibility

Unlock unknown-unknowns - from the kernel through userspace into high-level code, across **multi-cloud workloads.**

Polyglot Visibility

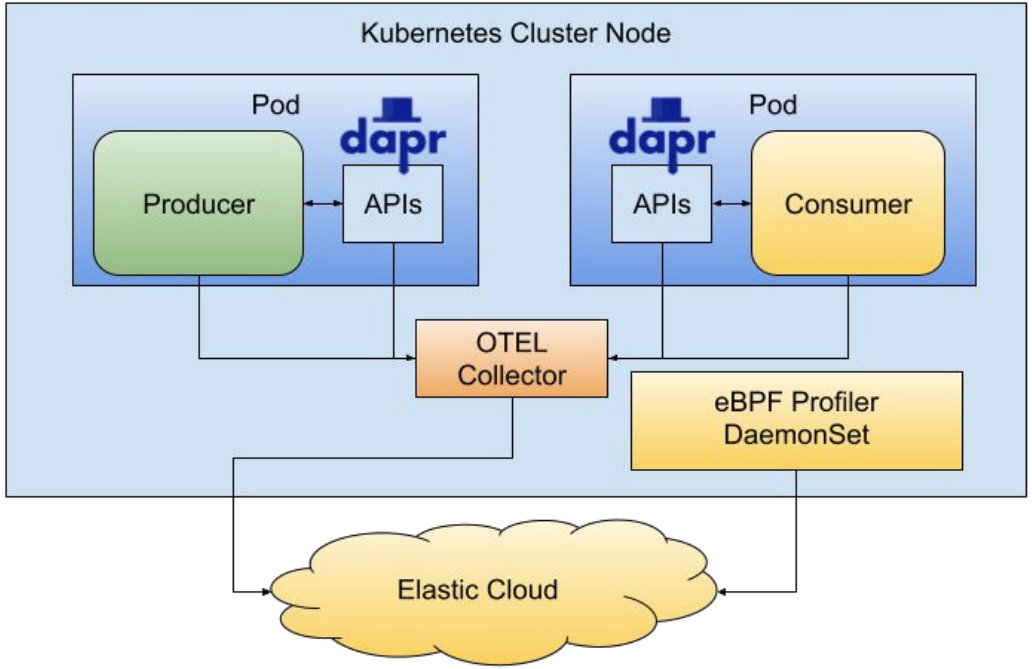
C/C++, Rust & Go (without debug symbols on host) PHP, Python, Java (or any JVM language), Ruby, DotNet, Perl & NodeJS.

Extremely Low Overhead

Continuous profiling in **production with negligible overhead**.

Typical case: < 1% CPU, ~250MB of RAM

Recap deployment



Demo #2

Profiling in production

What's ahead

- Stabilization of the profiling signal (OTLP)
- Stabilization of profiling support in the OTEL-collector
- Standardization of trace - profiling correlation
- More than CPU-profiling (e.g. IO, page-faults,etc)

… and much more!

Thanks!

Jonas Kunz (Jonas Kunz @ CNCF-Slack)

Mauricio Salatino @diagridio @daprdev @salaboy