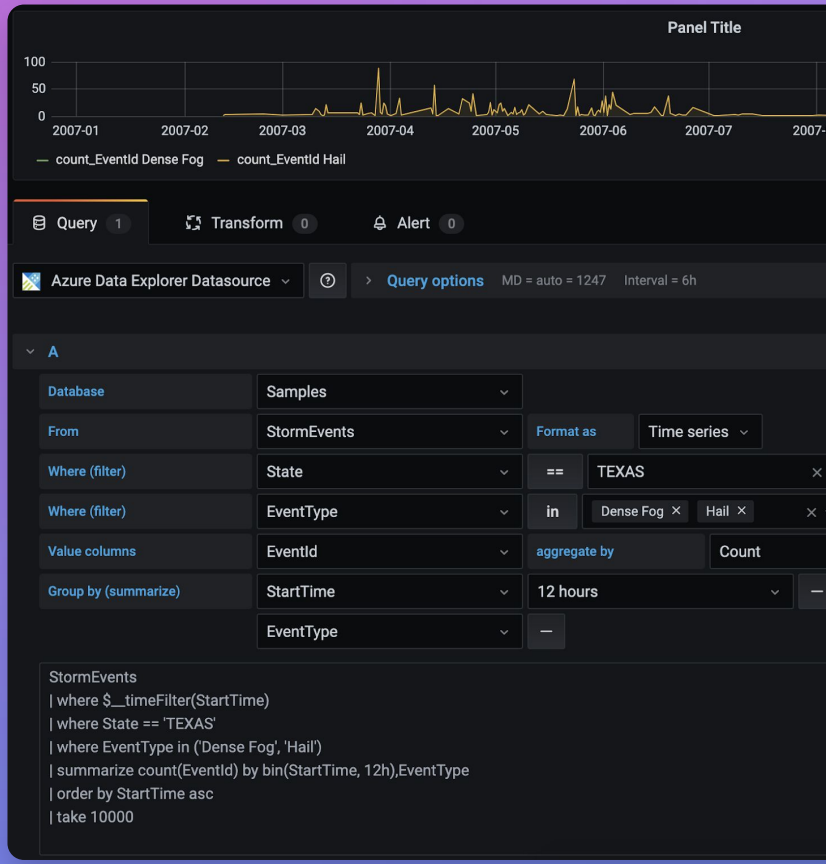
Grafana Labs

OpenTelemetry Collector deep dive

Juraci Paixão Kröhling
Software engineer
@jpkrohling



Presenter

Juraci Paixão Kröhling
Software engineer

Agenda

The basics

- What's OpenTelemetry
- What's OpenTelemetry Collector
- Other related projects: contrib and builder

Deployment patterns

- General purpose patterns (basic, normalizer, per-signal)
- Patterns for Kubernetes (daemonsets, sidecars)
- Enterprise patterns (multi-cluster, multitenant, load balancing)

Advanced topics

- Assembling your own distribution
- Extending with your components

Questions and answers

What's OpenTelemetry

OpenTelemetry is a collection of tools, APIs, and SDKs. Use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) to help you analyze your software's performance and behavior.

Source: https://opentelemetry.io/

Standards, specifications, and conventions

Client instrumentation APIs and SDKs

Middleware

Standards, specifications, and conventions

OpenTelemetry API / SDK

- For people looking to implement APIs and SDKs

Semantic conventions

- Which metadata to include in which operations

OpenTelemetry Line Protocol

- Interface description language (IDL), specifying how data should look like and what endpoints should implement

Client instrumentation APIs and SDKs

OpenTelemetry API

- What you use on a daily basis to instrument your service

OpenTelemetry SDK

- What to do with the instrumentation: how to create the data, buffer, send out

Instrumentation libraries

- Libraries that will hook into parts of your stack and instrument it


Middleware

What's OpenTelemetry Collector

Vendor-agnostic way to receive, process and export telemetry data.

Source: https://opentelemetry.io/docs/collector/

OpenTelemetry Collector conceptual architecture

OpenTelemetry Collector - related projects

Contrib

- Where non-core components reside, such as vendor-specific ones

Builder

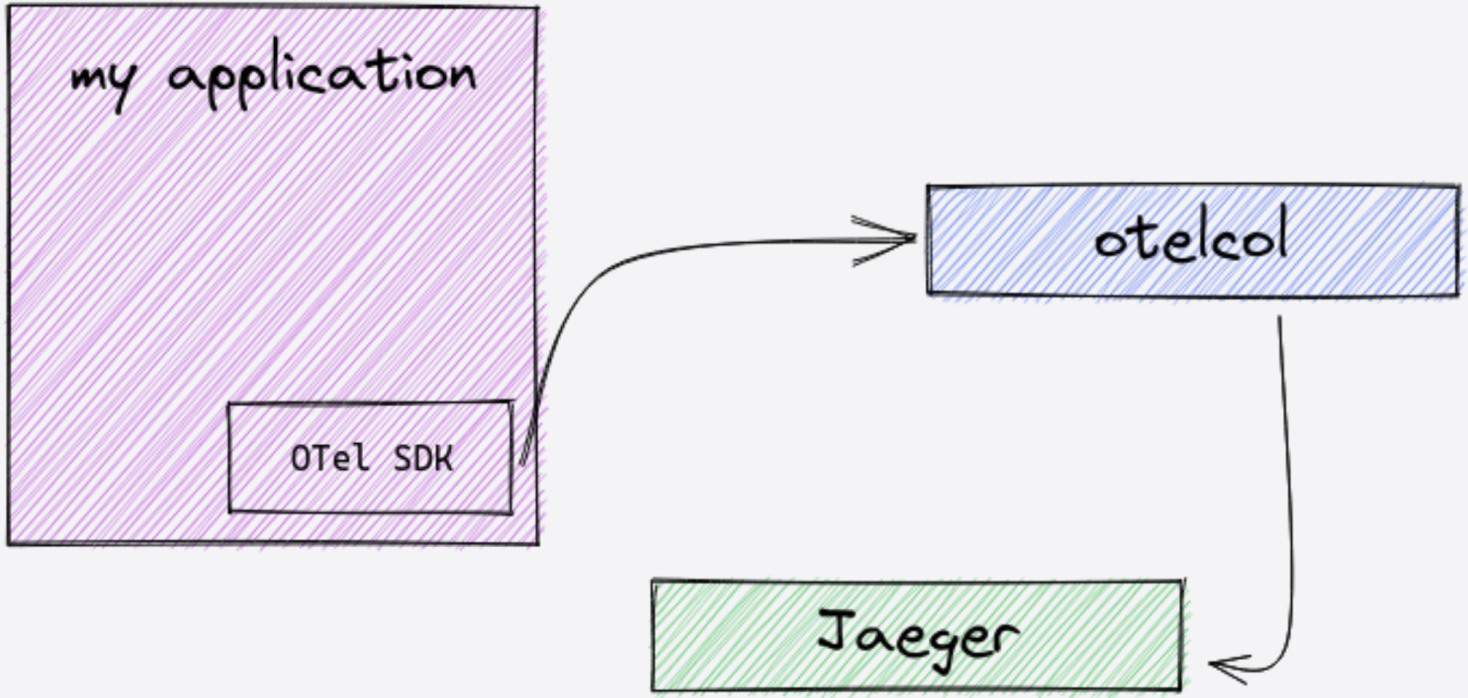
- Helper CLI tool to build OpenTelemetry Collector distributions

Operator

- Kubernetes operator managing OpenTelemetry Collector instances

Patterns!

Pattern #1 - Basic I



Pattern #1 - Basic I

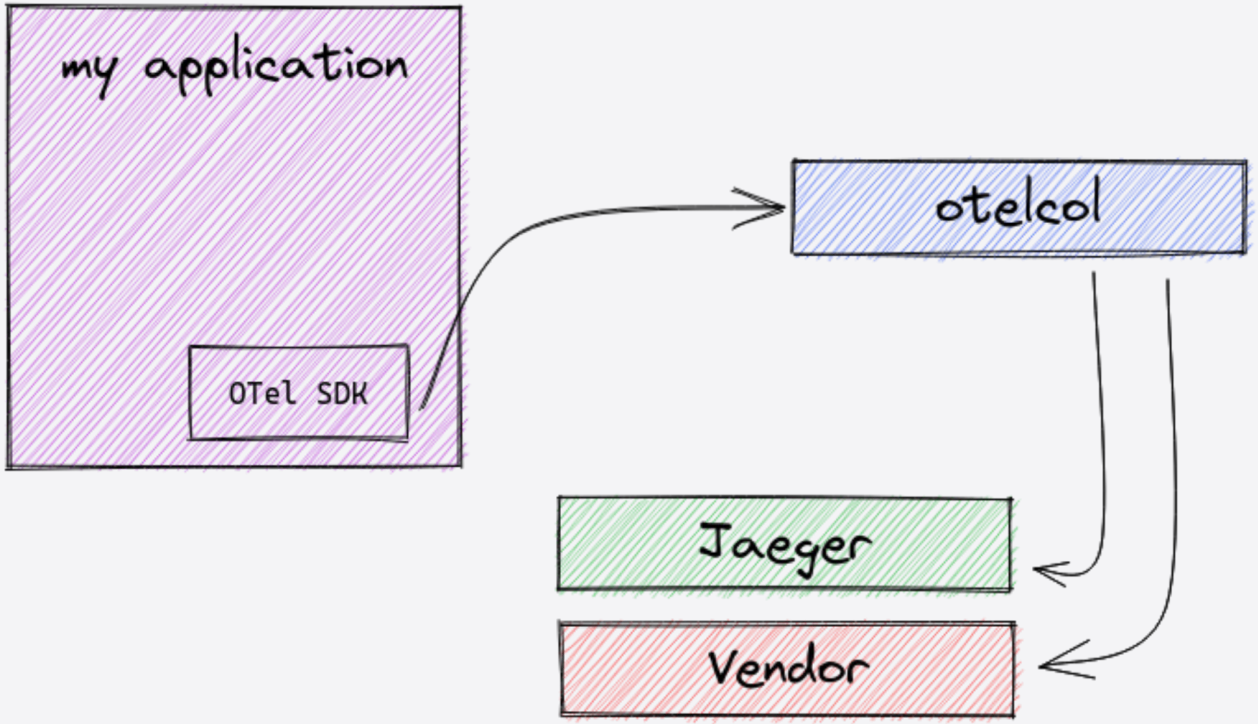
✅ Good for:

- Abstracting where to actually send the telemetry data
- Doing extra-processing between your workload and the telemetry backend

🚨 Avoid when:

- Well, when you don’t need an extra processing layer, every extra hop is a chance for things to go wrong 🐶

Pattern #1 - Basic II - Fanout



@jpkrohling

Pattern #1 - Basic II - Fanout

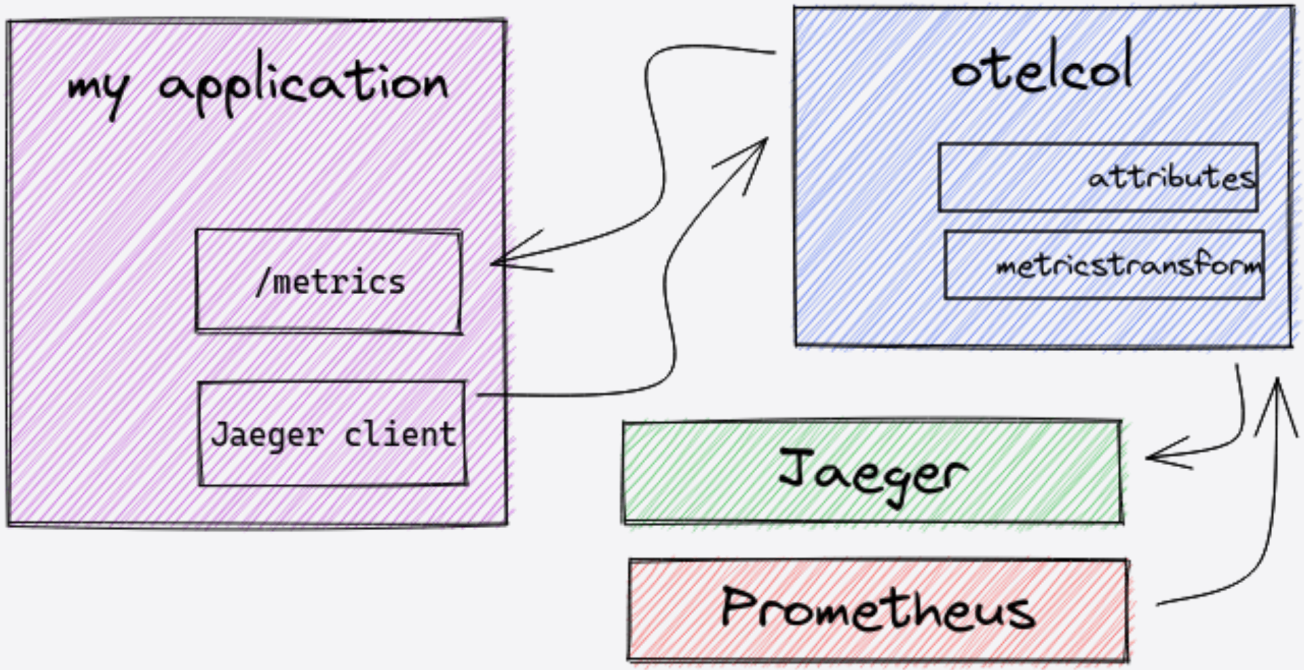
✅ Good for:

- Trying out different open source solutions and/or vendors
- Retaining data ownership even when your main observability tool is a SaaS

🚨 Avoid when:

- Processing tons of data: be conscious of the costs 💸

Pattern #2 - Normalizer



Pattern #2 - Normalizer

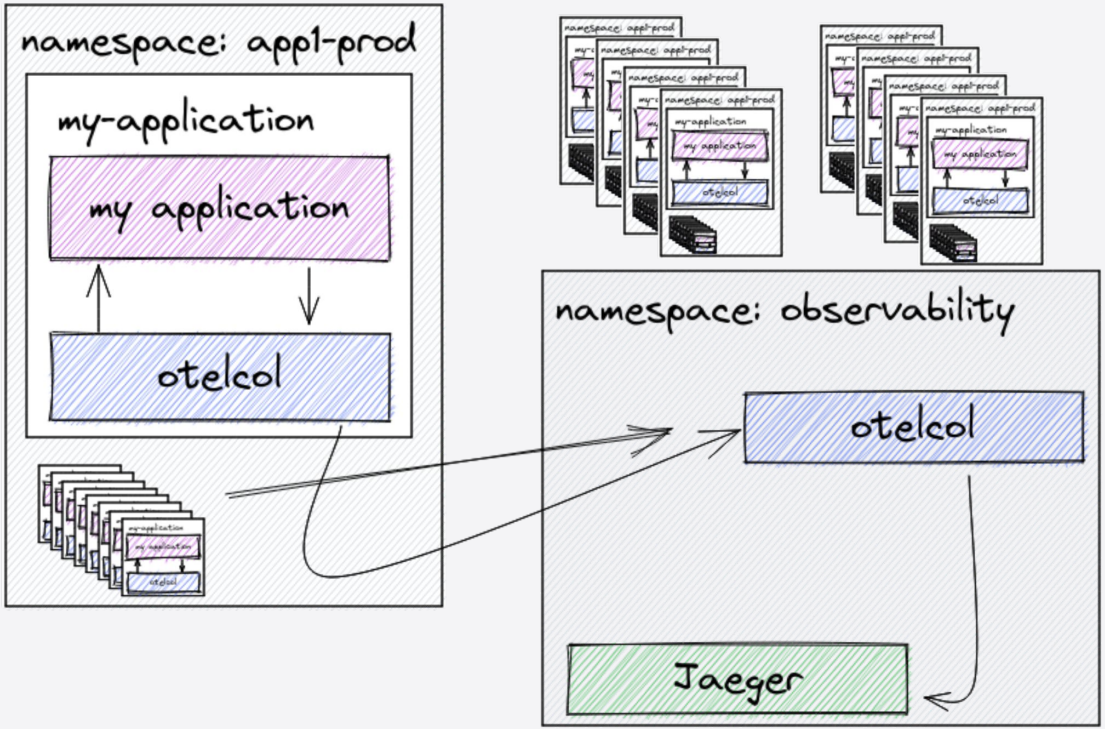
✅ Good for:

- Ensuring that different data points have the same semantics for the same things
- It's hard or undesirable to fix the problem at the source

🚨 Avoid when:

- You have too many things to normalize. It might be better to try to 🔧 fix the problem at the source

Pattern #3 - Kubernetes - Sidecars



Pattern #3 - Kubernetes - Sidecars

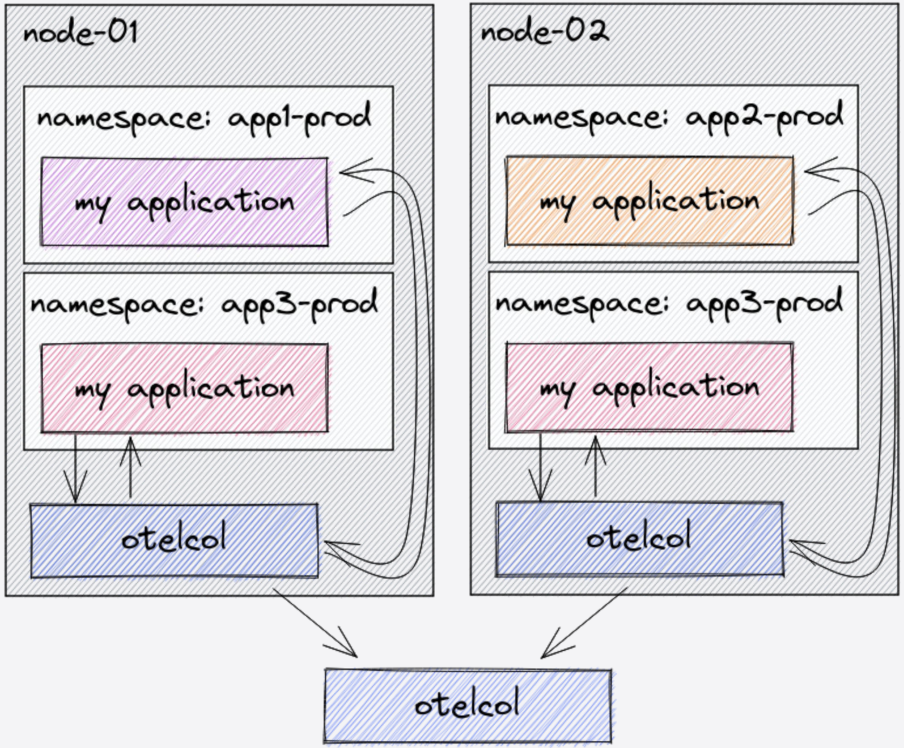
✅ Good for:

- Quickly offloading telemetry data from your application to a local process
- Fine-grained control over the configuration for each PodSpec or namespace
- Client-side load balancing is better when there are multiple of clients, especially for long-lived connections (HTTP/2, gRPC, Thrift, ...)

🚨 Avoid when:

- The overhead is not acceptable, as each sidecar needs at least ~20MiB of RAM
- You can’t use something like the operator to manage the configs

Pattern #3 - Kubernetes - DaemonSets



Pattern #3 - Kubernetes - DaemonSets

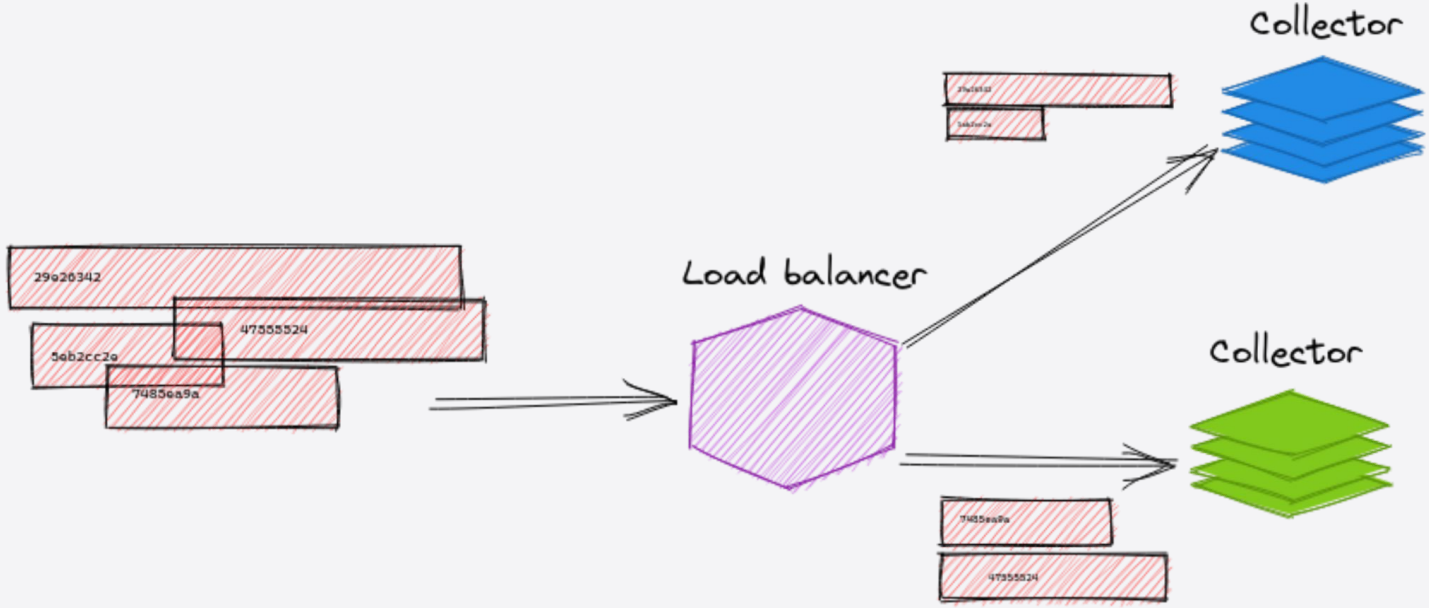
✅ Good for:

- Quickly offloading telemetry data from your application to a local process
- Less collector instances mean less maintenance and runtime overhead

🚨 Avoid when:

- You need multi-tenancy
- It's not acceptable to lose telemetry data for all pods on a node in case of a 💥 crash with the local collector

Pattern #4 - Load balancing



Pattern #4 - Load balancing

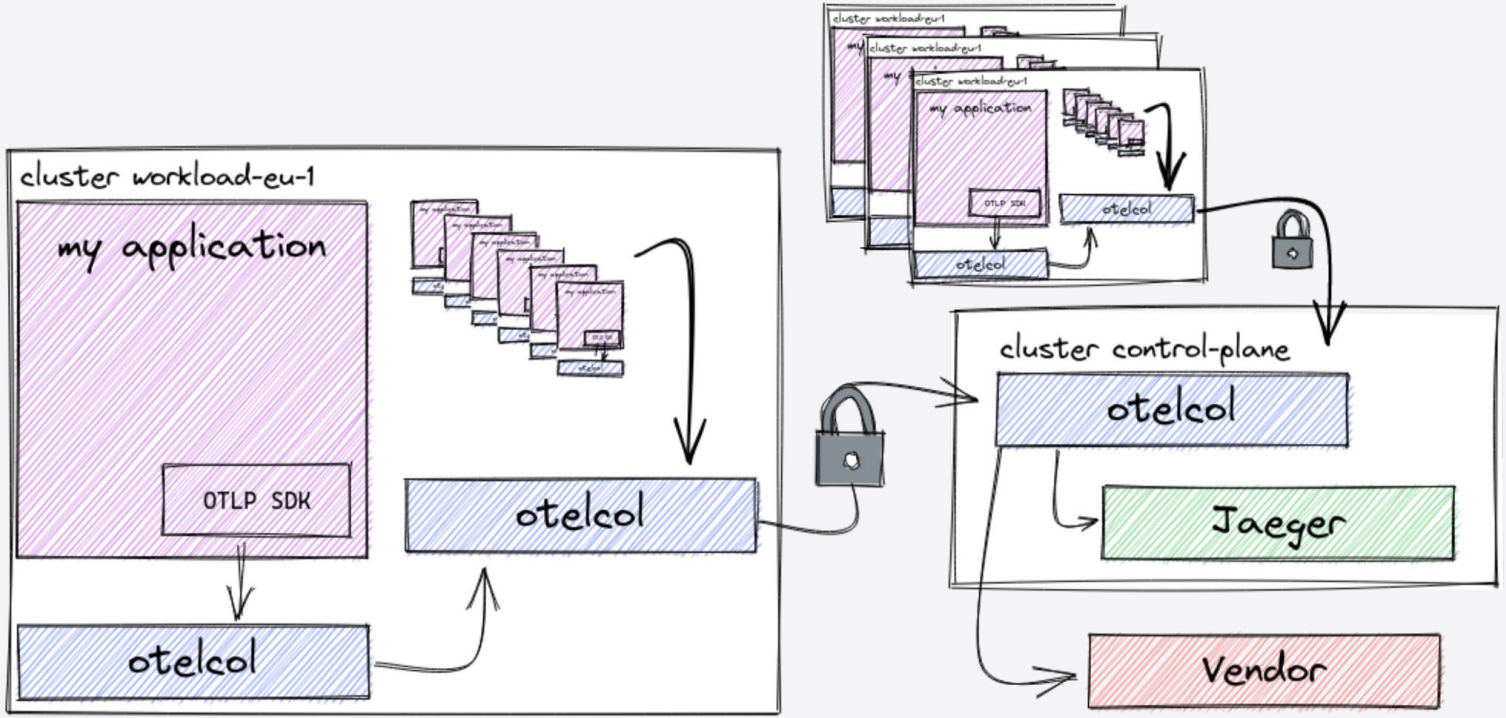
✅ Good for:

- Load balancing whole traces to collectors that need a complete view of the trace: span metrics processor, tail-based sampling, ...

🚨 Avoid when:

- You just need a simple load balancing, without caring about the trace ID at all. For that, use a regular HTTP/2 or gRPC load balancer.

Pattern #5 - Multi-cluster



Pattern #5 - Multi-cluster

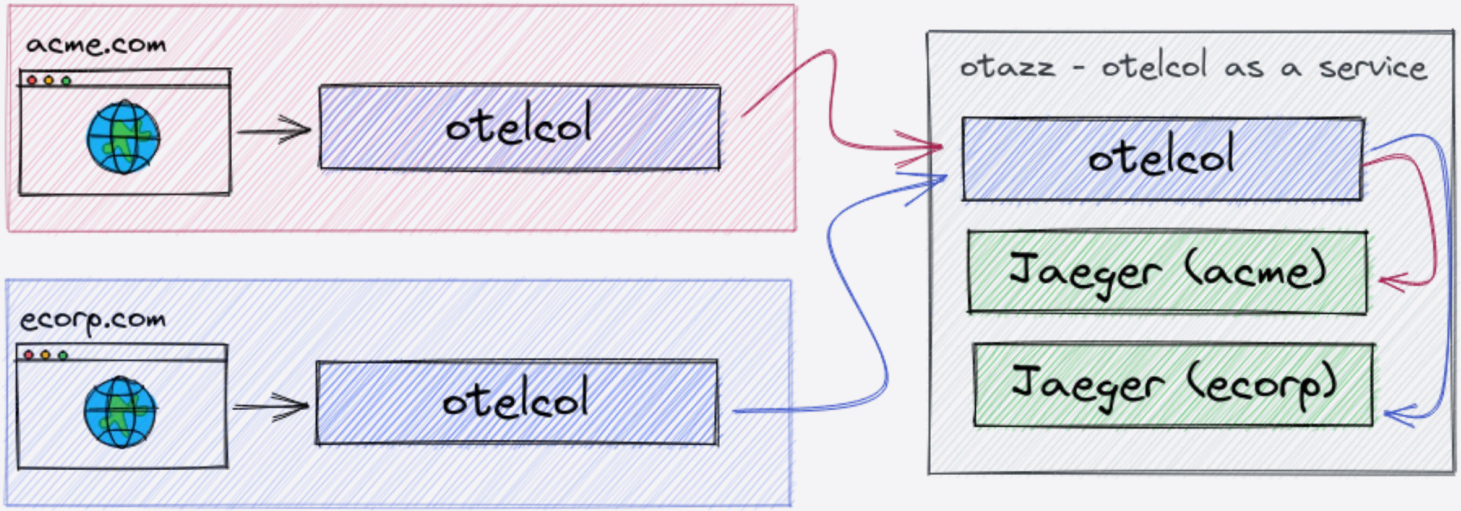
✅ Good for:

- Centralizing your telemetry data collection across clusters
- Running business analytics on all of your telemetry data

🚨 Avoid when:

- You can have your control plane to query data directly on individual clusters
- Networking costs are a concern

Pattern #6 - Multitenant



Pattern #6 - Multitenant

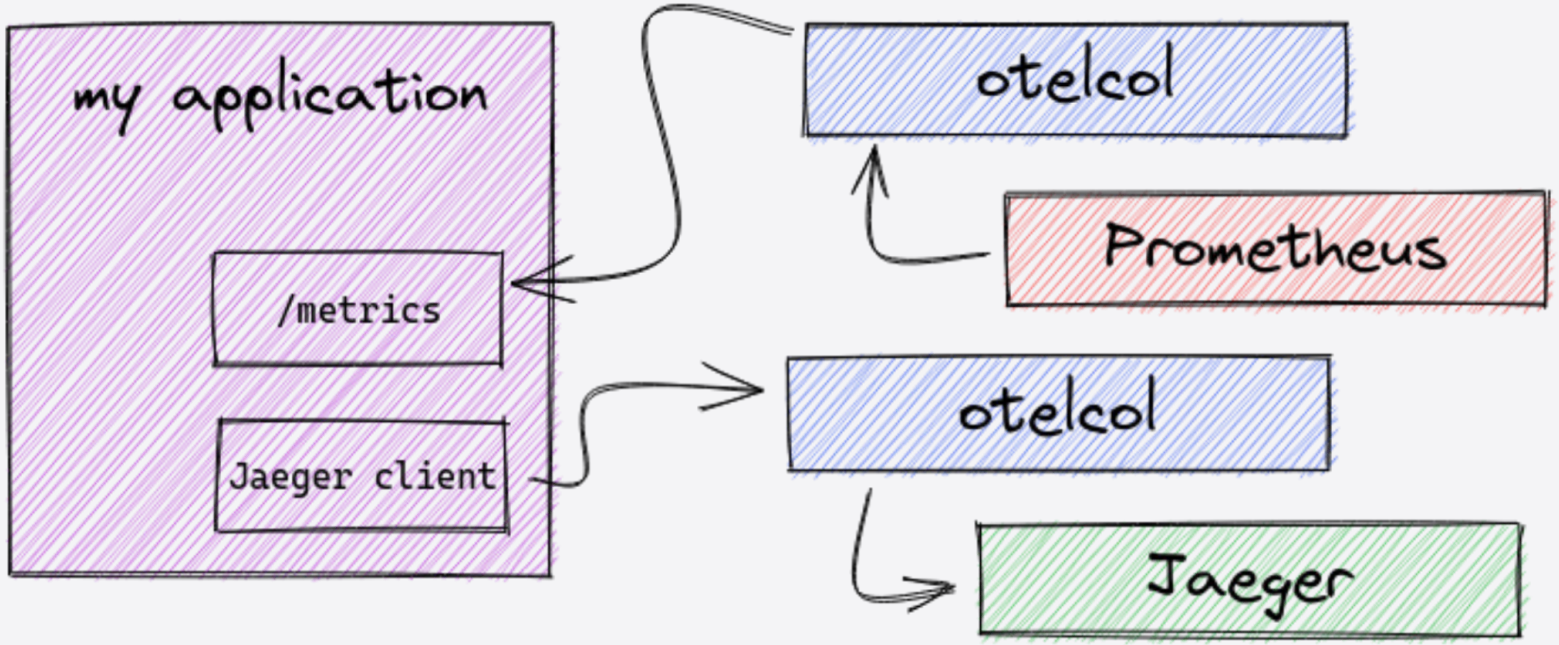
✅ Good for:

- Small deployments, where a central collector processes all the telemetry data for all tenants
- Central teams to handle telemetry backends for multiple departments

🚨 Avoid when:

- You can have one entrypoint per tenant, avoiding a single point of failure

Pattern #7 - Per signal



Pattern #7 - Per signal

✅ Good for:

- Isolating failures on production environments

🚨 Avoid when:

- You just need a simple deployment for your local dev or staging environments

Customizing OpenTelemetry Collector

Building your own distribution

1. manifest.yaml
2. opentelemetry-collector-builder
3. ???
4. Profit!

Building a component

- Config
- The component code, implementing one or more interfaces
- Factory

Building a processor

- Bootstrap go module
- Create a config.go
- Create a processor.go
- Add the processor logic
- Create a factory.go
- Bonus points: metrics.go

Key takeaways

Key takeaways

- OpenTelemetry has different subprojects in different areas
- The collector works as a middleware, abstracting the telemetry backends from your workloads
- It has tons of components for you to experiment with
- Mix and match collector instances with potentially different configurations
- Use the patterns from this presentation to derive your own patterns
- Extending the collector isn’t that hard!
- Building your own distribution might be a good idea depending on your use cases

Grafana Labs

Thanks for attending!

Have more questions?

@jpkrohling at twitter and github

Get involved:

#otel-collector
(CNCF Slack)

open-telemetry/
opentelemetry-collector