


MicroProfile Telemetry



whoami

- Martin Štefanko
- Software engineer, Red Hat
- MicroProfile committer
- Microservices enthusiast
-  @xstefank



OpenTelemetry

- OpenTracing and OpenCensus merger (May 2019)
- OpenTelemetry's Mission: to enable effective observability by making high-quality, portable telemetry ubiquitous.
- CNCF incubating project



OpenTelemetry

- Components:
 - Cross-language specification
 - Tools to collect, transform, and export telemetry data
 - Per-language SDKs
 - Automatic instrumentation and contrib packages

Microservices

App Code

OTel Auto. Inst.

OTel API

OTel SDK

OTLP

3rd party
service

OTel
Collector

Time Series
Databases

Trace
Databases

Observability
Frontends
& APIs

Column
Stores

Kubernetes

OTLP

L7 Proxy

OTLP



OTLP

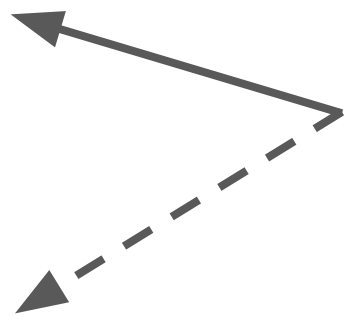
Shared
Infra

Managed DBs

APIs

Client Instrumentation

OpenTelemetry

- Four main categories (signals)
 - Traces
 - Metrics
 - Logs
 - Baggage
- 
- The diagram illustrates the relationship between OpenTelemetry signals and MicroProfile Telemetry Tracing. It features a solid arrow pointing from the 'Traces' category to 'MicroProfile Telemetry Tracing' and a dashed arrow pointing from 'Baggage' to the same target.
- ```
graph LR; Traces --> MPTT[MicroProfile Telemetry Tracing]; Baggage -.-> MPTT;
```

# MP Telemetry Tracing

Instrumentation types:

- Automatic
- Agent
- Manual

# Automatic Instrumentation

- JAX-RS
  - Server
  - Client
- MP REST Client



# Agent Instrumentation

- Java Agent JAR added to the existing application
- If provided by implementation then it must adhere to the OTel configuration specification

# Manual Instrumentation

```
@ApplicationScoped
class SpanBean {

 @WithSpan
 void span() {
 }

}
```

# Manual Instrumentation

```
@ApplicationScoped
class SpanBean {

 @WithSpan("name")
 void span() {
 }

}
```

# Manual Instrumentation

```
@ApplicationScoped
class SpanBean {

 @WithSpan(kind = SpanKind.SERVER)
 void span() {
 }

}
```

# Manual Instrumentation

```
@ApplicationScoped
class SpanBean {

 @WithSpan
 void span(@SpanAttribute(value = "arg") String arg)
{ }

}
```

# Manual Instrumentation

```
// import io.opentelemetry.api.trace.Tracer;

@ApplicationScoped
class SpanBean {

 @Inject
 Tracer tracer;

}
```

# Manual Instrumentation

```
Span span = tracer.spanBuilder("custom.span")
 .setSpanKind(SpanKind.INTERNAL)
 .setParent(Context.current().with(this.span))
 .setAttribute("attr", "value")
 .startSpan();

// execute traced task

span.end();
```

\*automatically added as child span to the spans started for JAX-RS resources

# CDI Injections

- `io.opentelemetry.api.OpenTelemetry`
- `io.opentelemetry.api.trace.Tracer`
- `io.opentelemetry.api.trace.Span`
- `io.opentelemetry.api.baggage.Baggage`



# Static methods

- `io.opentelemetry.api.GlobalOpenTelemetry.get()`
- `io.opentelemetry.api.trace.Span.current()`
- `io.opentelemetry.api.baggage.Baggage.current()`

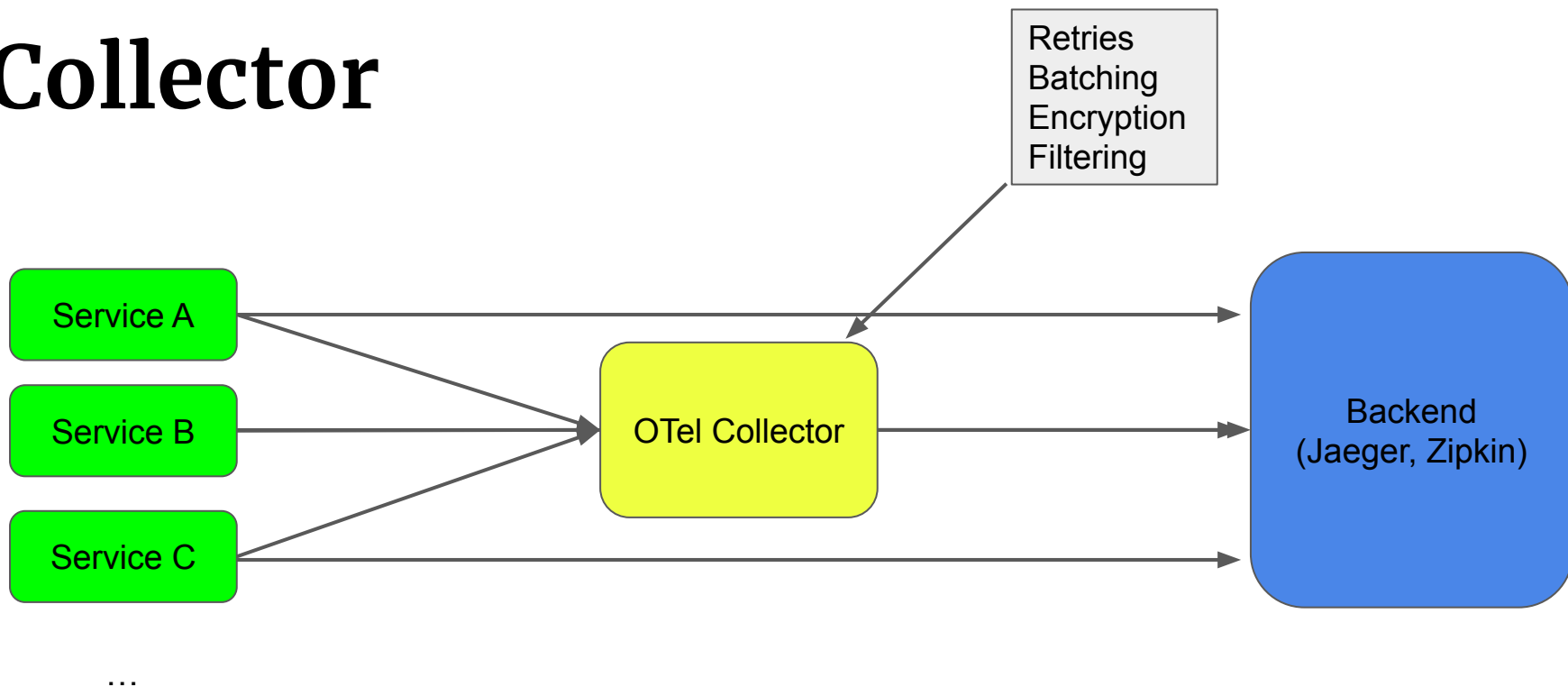
# Enabling MP Telemetry Tracing

- Disabled by default
- `otel.sdk.disabled=false`

# MP Telemetry vs MP OpenTracing

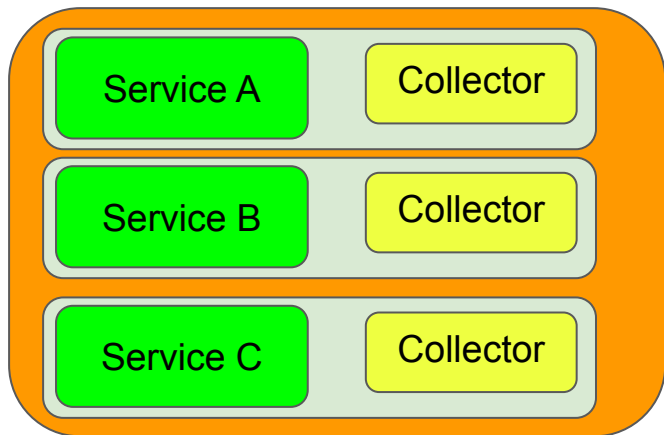
- Uncompatible
- Different APIs
- No MP API (@Traced annotation)
- No MP specific config

# Collector

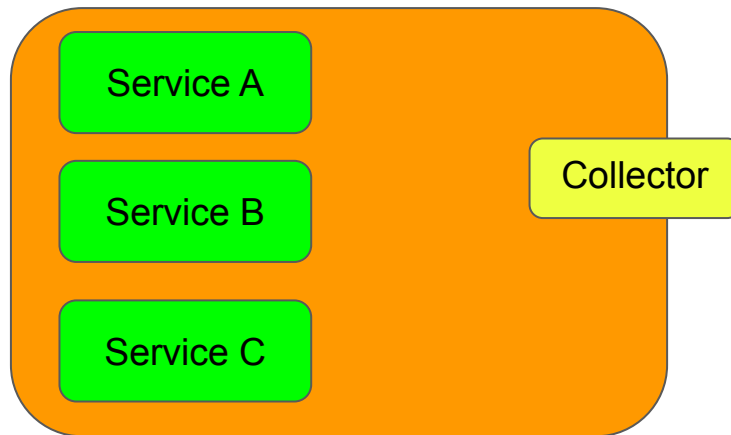


# Collector

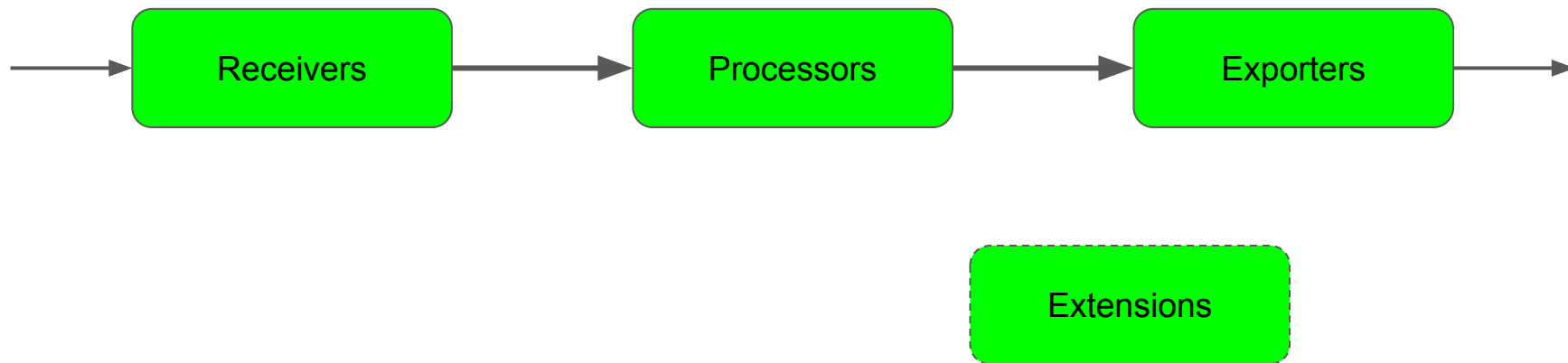
## Agent





## Gateway



# Collector



# Thank you

-  @xstefank
-  xstefank
- mstefank@redhat.com



## Code

<https://github.com/xstefank/mp-telemetry-demo>

## Slides

<https://bit.ly/3VMQLnV>

# Resources

- <https://opentelemetry.io/docs/>