Auto-generated Coherent Observability

Arthur Sens
SWE @ Grafana Labs

Dominik Süß
DevEx @ Grafana Labs

Prometheus

Where's the error?

```go
func main() {
    flag.Parse()

    reg := prometheus.NewRegistry()

    requestDurations := prometheus.NewHistogramVec(prometheus.HistogramOpts{
        Name:        "http_request_duration_seconds",
        Help:        "A histogram of the HTTP request durations in seconds.",
        Buckets:     prometheus.ExponentialBuckets(0.1, 1.5, 5),
    }, []string{"method", "code"})

    reg.MustRegister(
        requestDurations,
    )

    requestDurations.WithLabelValues("200", "GET", "/login").Observe(1.2)

    http.Handle("/metrics", promhttp.HandlerFor(reg, promhttp.HandlerOpts{Registry: reg}))
    log.Fatal(http.ListenAndServe(*addr, nil))
}
```

@ArthurSens
@theSuess

Where's the error?

```
func main() {
    flag.Parse()

    reg := prometheus.NewRegistry()

    requestDurations := prometheus.NewHistogramVec(prometheus.HistogramOpts{
        Name:        "http_request_duration_seconds",
        Help:        "A histogram of the HTTP request durations in seconds.",
        Buckets:     prometheus.ExponentialBuckets(0.1, 1.5, 5),
    }, []string{"method", "code"})

    reg.MustRegister(
        requestDurations,
    )

    requestDurations.WithLabelValues("200", "GET", "/login").Observe(1.2)

    http.Handle("/metrics", promhttp.HandlerFor(reg, promhttp.HandlerOpts{Registry: reg}))
    log.Fatal(http.ListenAndServe(*addr, nil))
}
```

@ArthurSens
@theSuess

Where's the error?

```
func main() {
    flag.Parse()

    reg := prometheus.NewRegistry()

    requestDurations := prometheus.NewHistogramVec(prometheus.HistogramOpts{
        Name:        "http_request_duration_seconds",
        Help:        "A histogram of the HTTP request durations in seconds.",
        Buckets:     prometheus.ExponentialBuckets(0.1, 1.5, 5),
    }, []string{"method", "code"})

    reg.MustRegister(
        requestDurations,
    )

    requestDurations.WithLabelValues("200", "GET", "/login").Observe(1.2)

    http.Handle("/metrics", promhttp.HandlerFor(reg, promhttp.HandlerOpts{Registry: reg}))
    log.Fatal(http.ListenAndServe(*addr, nil))
}
```

@ArthurSens
@theSuess

Where's the error?

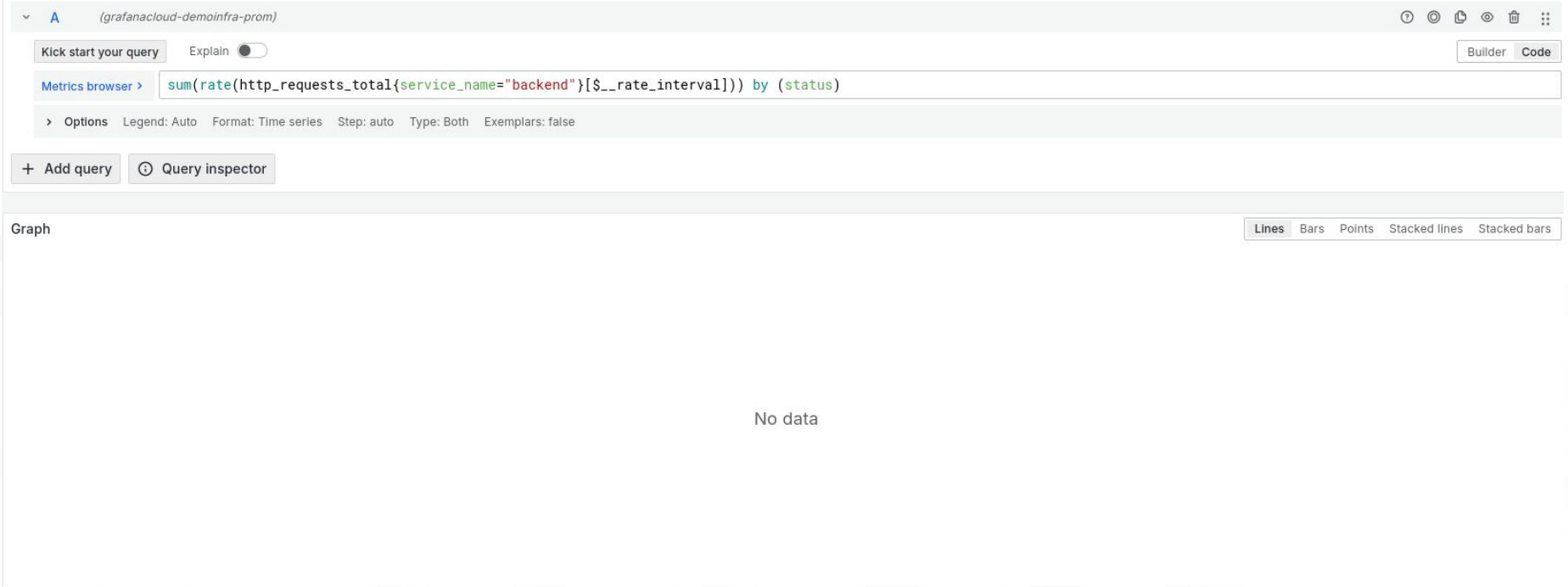
```
MacBook-Pro-de-Arthur:client_golang arthursens$ go run examples/simple/main.go
panic: inconsistent label cardinality: expected 2 label values but got 3 in []string{"200", "GET", "/login"}
```

@ArthurSens
@theSuess

Telemetry Sprawl

@ArthurSens
@theSuess

Telemetry Sprawl



@ArthurSens
@theSuess

Semantic Conventions

- Agreed upon names for common signals
- Community Maintained
- Many *namespaces* already populated
  - Networking, HTTP, Hardware, CICD,...

Example

HTTP server

Metric: `http.server.request.duration`

This metric is required.

When this metric is reported alongside an HTTP server span, the metric value SHOULD be th

This metric SHOULD be specified with `ExplicitBucketBoundaries` of

Name	Instru			tabi
`http.server.reques`				table

			Examples	Requirement Level	Stability
		me	`GET` ; `POST` ; `HEAD`	`Required`	stable
	string	The URI scheme component identifying the used protocol. [2]	`http` ; `https`	`Required`	stable
`http.route`	string	The matched route, that is, the path template in the format used by the respective server framework. [4]	`/users/:userID?` ; `{controller}/{action}/{id?}`	`Conditionally Required` If and only if it's available	stable

Build your own!

@ArthurSens
@theSuess

My first conventions

📚 Library Semantic Conventions

Metrics

Book count

`library.books.total`	Number of books known to the library instance	Stable

Attributes

`library.location`	Location of the library currently holding the book	Required	Stable
`library.book.status`	Availability status of the book	Required	Stable

@ArthurSens
@theSuess

Building your own conventions is hard!

- Cross-referencing labels
- Keeping documentation up to date
- Ownership process
- Ensuring teams adopt this

```
MacBook-Pro-de-Arthur:client_golang arthursens$ go run examples/simple/main.go
panic: inconsistent label cardinality: expected 2 label values but got 3 in []string{"200", "GET", "/login"}
```

@ArthurSens
@theSuess

OpenTelemetry Weaver

Observability by Design

Treat your telemetry like a public API

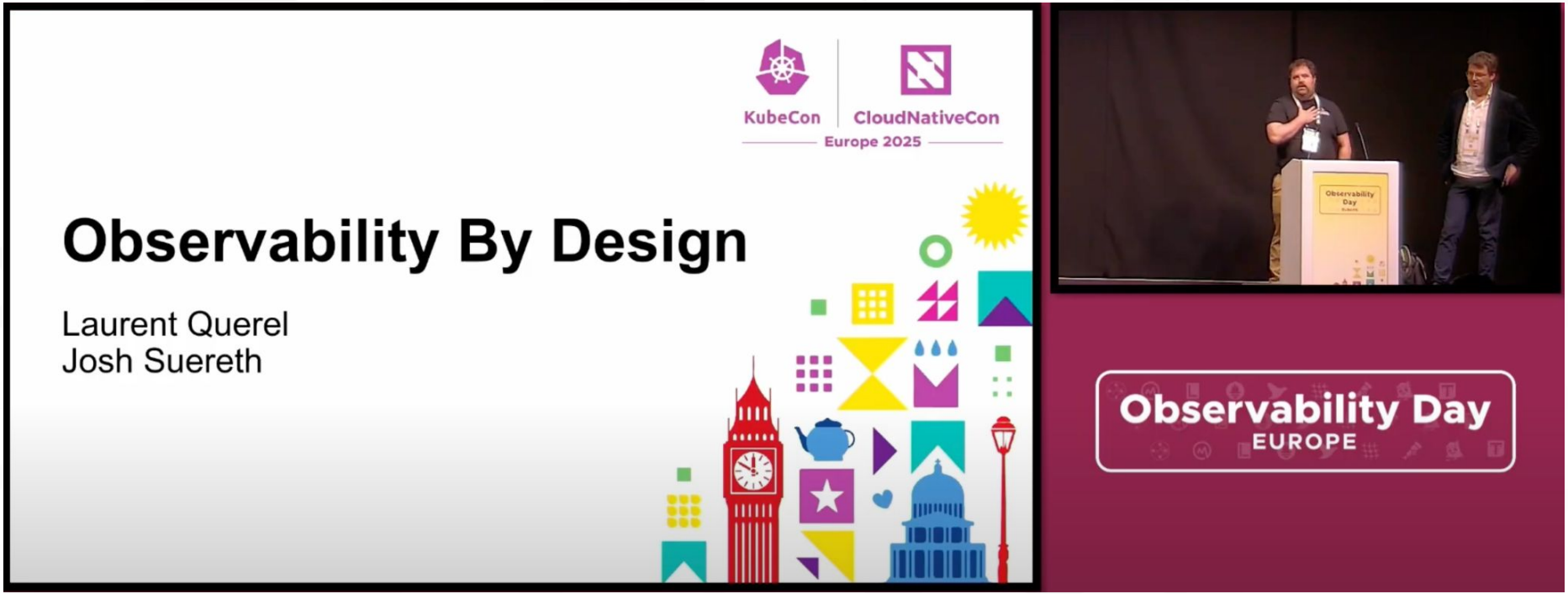
@ArthurSens
@theSuess

Weaver

- Semantic convention tooling
- Code generation
- Policy validation
- Live checking

@ArthurSens
@theSuess

OpenTelemetry Schemas



@ArthurSens
@theSuess

Weaver

- Semantic convention tooling
- Code generation -> OTel SDK / Prometheus SDK / Dashboards
- Policy validation
- Live checking

Community demand

prometheus / client_golang

Code | Issues 85 | Pull requests 29 | Discussions | Actions

Type-safe labels support? #1599

Open | #1598

26 18

amberpixels opened on Aug 28, 2024 · edited by amberpixels — Edits

Context

Currently when using a metric we can provide labels either with an ordered string list, or via `prometheus.Labels` map. Both seems to be very dangerous: chance of misorder, misspell, extra/missing values. Also it's not convenient, that you do not know which labels you can use until you reach the code of metric creation.

It's especially dangerous when using `promauto` as it fails on every issue with label.

Demo Time

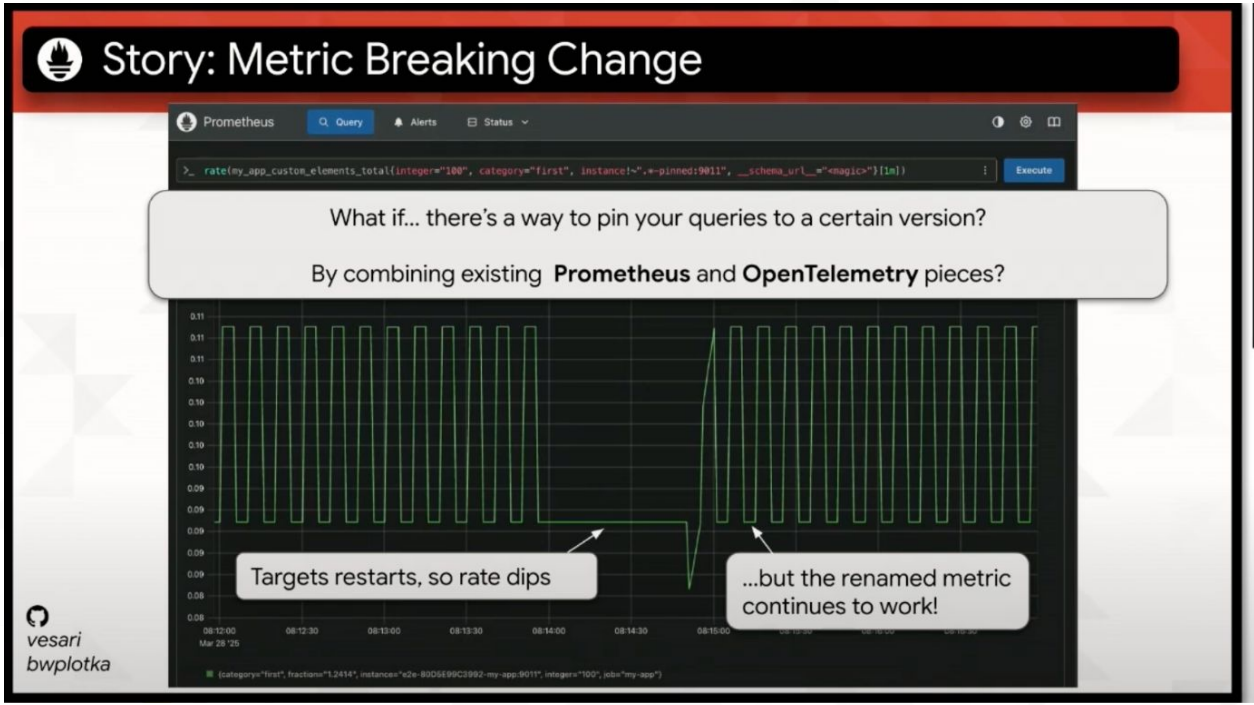
Prometheus

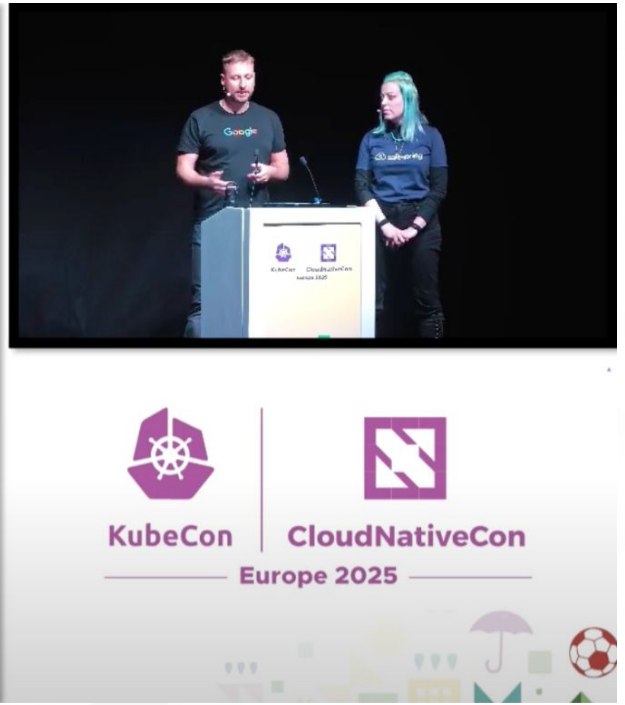
Building your own conventions is hard!

- Cross-referencing labels
  - id references
- Keeping documentation up to date
  - Generate Markdown
- Ownership process
  - Keeping conventions in a repo
- Ensuring teams adopt this
  - Good libraries / live checking

@ArthurSens
@theSuess

OpenTelemetry Schemas





@ArthurSens
@theSuess

Future vision

- Prometheus SDKs providing the templates and automations.
- Prometheus instrumented via auto-generated instrumentation.
- Prometheus exporters, too!
- Mixins replaced with Schemas
    - Each dashboard vendor to provide their automation on top of Schemas.