Grafana Labs

Philosophy of Observability

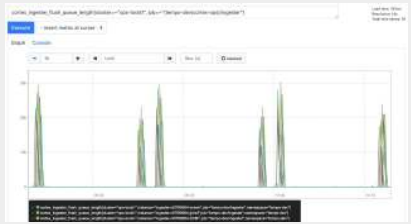
Leveraging Cloud-Native Technologies

Richard “RichiH” Hartmann

Today's reality:
Disparate systems. Disparate data.

ORACLE

APPDYNAMICS



elasticsearch

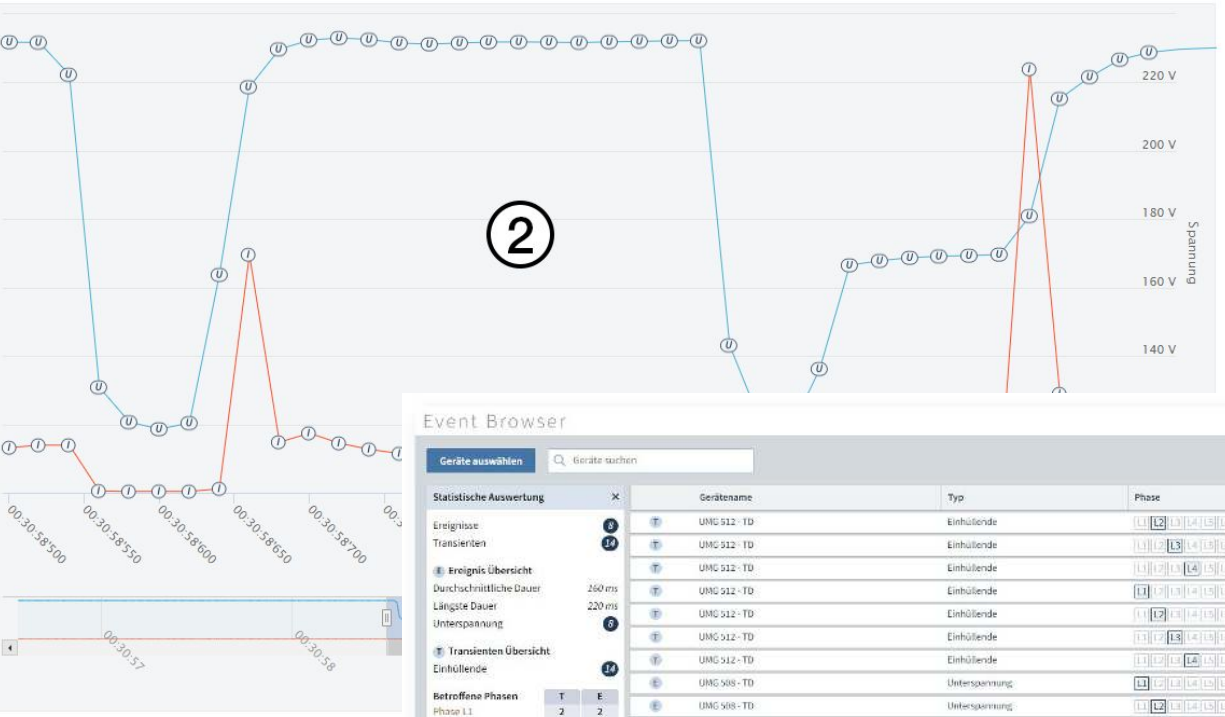
splunk>

Grafana

Back to the basics

Let’s rethink this

How humanity deals with data



②

Event Browser

Geräte auswählen

Geräte suchen

19.12.2019 - 19.12.2019
00:00 - 23:59

Statistische Auswertung

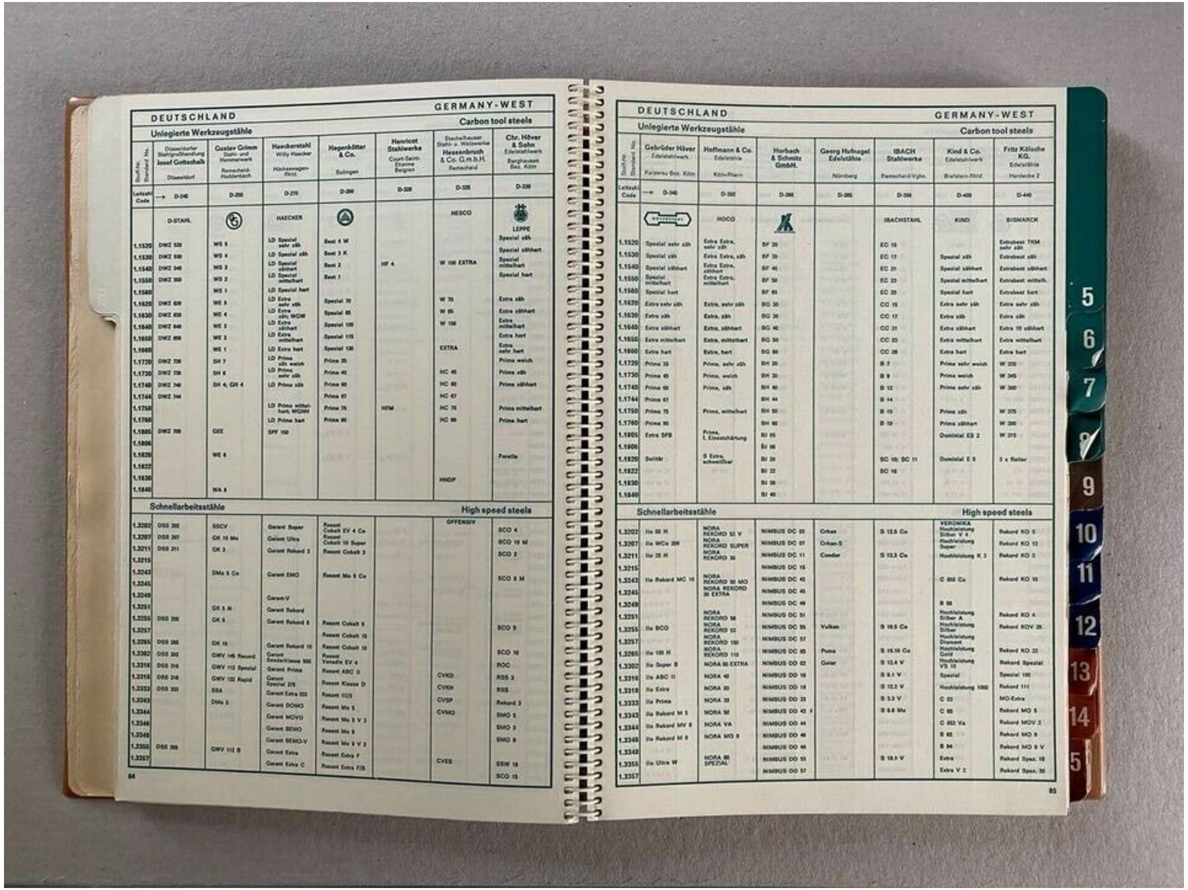
Ereignisse	8
Transienten	14
Ereignis Übersicht	
Durchschnittliche Dauer	160 ms
Längste Dauer	220 ms
Unterspannung	8
Transienten Übersicht	
Einhüllende	14

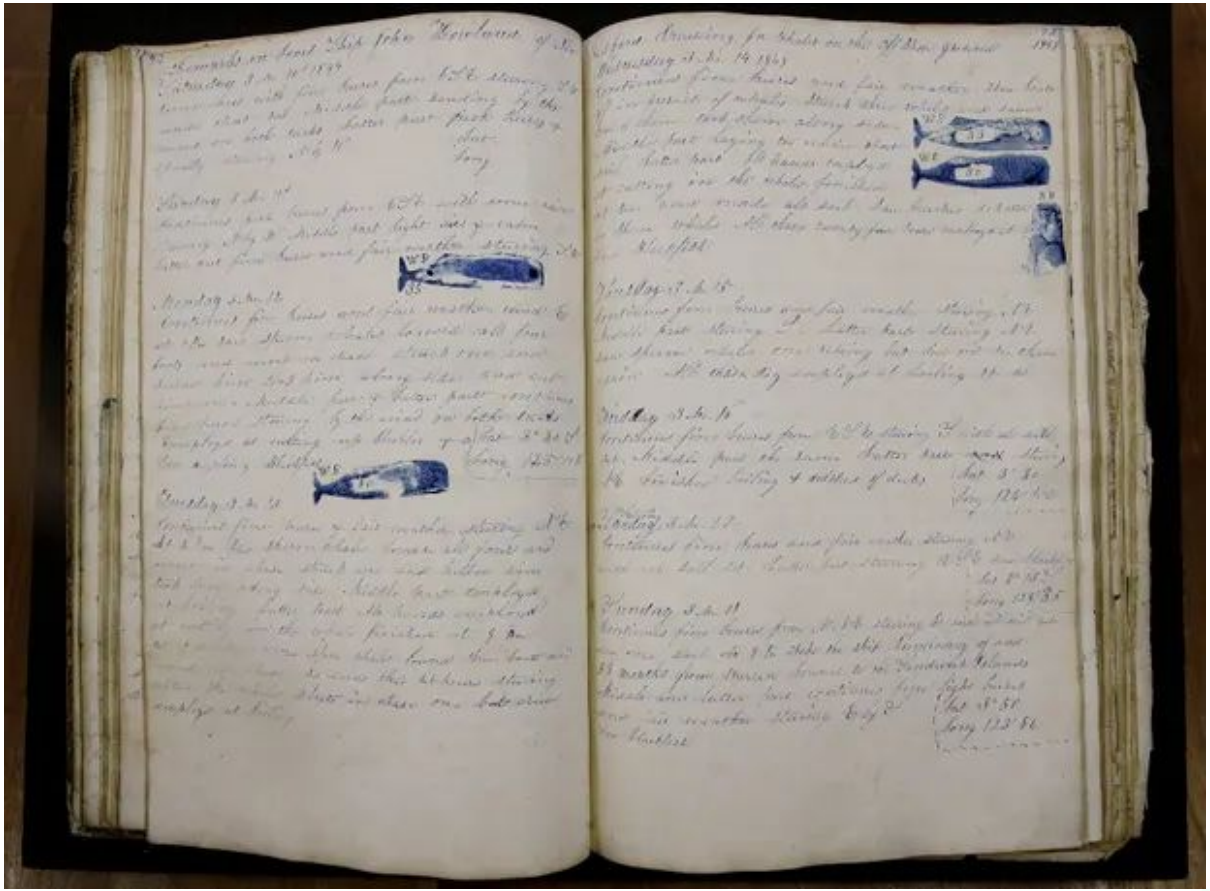
Betroffene Phasen	T	E
Phase L1	2	2
Phase L2	4	2
Phase L3	4	2
Phase L4	4	2

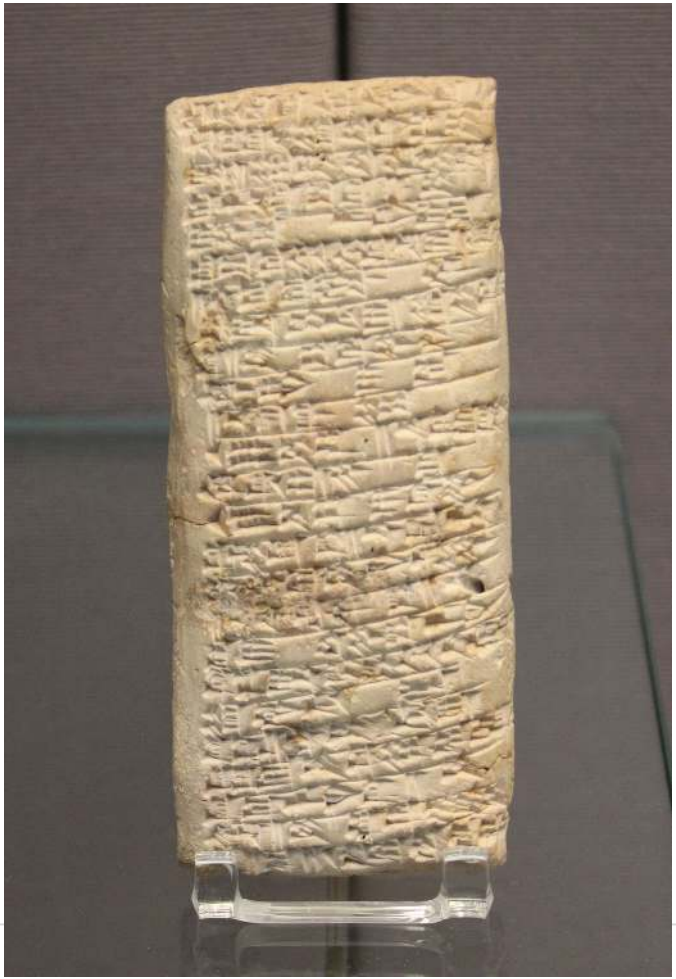
	Gerätename	Typ	Phase	Start	Ende	Dauer	Wert	
T	UMG 512 - TD	Einhüllende	L2	19.12.2019 01:15:40'663	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L3	19.12.2019 01:15:40'663	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L4	19.12.2019 01:15:40'663	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L1	19.12.2019 01:15:39'345	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L2	19.12.2019 01:15:39'345	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L3	19.12.2019 01:15:39'345	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L4	19.12.2019 01:15:39'345	---	---	---	Analysieren
E	UMG 508 - TD	Unterspannung	L1	19.12.2019 00:30:59'979	19.12.2019 00:30:59'199	220 ms	117.836 V (MIN)	Analysieren
E	UMG 508 - TD	Unterspannung	L2	19.12.2019 00:30:59'979	19.12.2019 00:30:59'199	220 ms	117.806 V (MIN)	Analysieren
E	UMG 508 - TD	Unterspannung	L3	19.12.2019 00:30:59'979	19.12.2019 00:30:59'199	220 ms	117.825 V (MIN)	Analysieren
E	UMG 508 - TD	Unterspannung	L4	19.12.2019 00:30:59'979	19.12.2019 00:30:59'199	220 ms	117.830 V (MIN)	Analysieren
E	UMG 508 - TD	Unterspannung	L1	19.12.2019 00:30:58'659	19.12.2019 00:30:58'659	200 ms	118.640 V (MIN)	Analysieren
E	UMG 508 - TD	Unterspannung	L2	19.12.2019 00:30:58'659	19.12.2019 00:30:58'659	200 ms	118.612 V (MIN)	Analysieren
E	UMG 508 - TD	Unterspannung	L3	19.12.2019 00:30:58'659	19.12.2019 00:30:58'659	200 ms	118.622 V (MIN)	Analysieren
E	UMG 508 - TD	Unterspannung	L4	19.12.2019 00:30:58'659	19.12.2019 00:30:58'659	200 ms	118.631 V (MIN)	Analysieren
T	UMG 512 - TD	Einhüllende	L2	19.12.2019 00:28:40'565	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L3	19.12.2019 00:28:40'565	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L4	19.12.2019 00:28:40'565	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L1	19.12.2019 00:28:40'130	---	---	---	Analysieren
T	UMG 512 - TD	Einhüllende	L2	19.12.2019 00:28:40'130	---	---	---	Analysieren

20 1

Grafana Labs


DEUTSCHLAND
GERMANY-WEST
Unlegierte Werkzeugstähle
Carbon tool steels
Schnellarbeitsstähle
High speed steels





Humanity has optimized detailed accounts into key events into numbers for millenia

Again and again and again

Observability & SRE

Or: Buzzwords, and their useful parts

Observability, the buzzword

- Cool new term, almost meaningless by now, what does it mean?
  - Pitfall alert: Cargo culting
  - It's about changing the behaviour, not about changing the name
- "Monitoring" has taken on a meaning of collecting, not using data
  - One extreme: Full text indexing
  - Other extreme: Data lake
- "Observability" is about enabling humans to understand complex systems
  - Ask new questions on the fly
  - Ask **why** it's not working instead of just knowing that it's not
- Terms such as "Observability 2.0", "Observability 3.0", and "Observability 4.0", are other examples of buzzwordiness

[...] observations are [...] approximations to the truth [...] this can be accomplished in no other way than by a suitable combination of more observations than the number absolutely requisite for the determination of the unknown quantities

Carl Friedrich Gauß, 1809

Observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs.

Rudolf Emil Kálmán, 1960

Complexity

- Fake complexity, a.k.a. bad design
  - Can be reduced
- Real, system-inherent complexity
  - Can be moved (monolith vs client-server vs microservices)
  - Must be compartmentalized (service boundaries)
  - Should be distilled meaningfully (observability...)
  - Can **not** be reduced

Services

- What's a service?
  - Compartmentalized complexity, with an interface
  - Different owners/teams
  - Contracts define interfaces
- Why "contract": Shared agreement which MUST NOT be broken
  - Internal and external customers rely on what you build and maintain
- Other common term: layer
  - The Internet would not exist without network layering
  - Enables innovation, parallelizes human engineering
- Other examples: CPUs, hard drive, compute nodes, your lunch

Cloud-native vs client-server vs mainframe vs...

- A mainframe application and a microservices fleet are fundamentally the same
  - You can *move* system-inherent complexity, but...
- Microservices broke up old service and system boundaries
  - Enabling horizontal scalability, arguably at the cost of vertical scalability
- Previous-generation tooling is designed to understand system complexity along existing service boundaries
  - Cloud native tooling is able to deal with this increased complexity
  - NB: This means previous-generation complexity is even easier to observe

SRE

- At its core: Align incentives across the org
  - Error budgets allow devs, ops, PMs, etc. to optimize for shared benefits
- Measure it!
  - SLI: Service Level Indicator: What you measure
  - SLO: Service Level Objective: What you need to hit
  - SLA: Service Level Agreement: When you need to pay
- Discern between different SLIs
  - Primary: service-relevant, for alerting
  - Secondary: informational, debugging, might be underlying's primary

Shared understanding

- Everyone uses the same tools & dashboards
  - Shared incentive to invest into tooling
  - Pooling of institutional system knowledge
  - Shared language & understanding of services

Alerting

- Customers care about services being up, not about individual components

Anything currently or imminently impacting customer service must be alerted upon

But nothing(!) else

Prometheus

Prometheus 101

- Inspired by Google’s Borgmon
- Time series database
- Rich ecosystem, 1,000s of instrumentations & exporters
- Cloud-native default

Time series

- Time series are recorded values which change over time
- Individual events are usually merged into counters and/or histograms
- Changing values are recorded as gauges
- Typical examples
  - Requests to a webserver (counter)
  - Temperatures in a datacenter (gauge)
  - Service latency (histograms)

Cloud-native default

- Kubernetes =~ Borg
- Prometheus =~ Borgmon
- Google couldn't have run Borg without Borgmon
- Kubernetes & Prometheus are designed and written with each other in mind

Prometheus 101

- Black-box monitoring: Looking at a service from the outside (Does the server answer to HTTP requests?)
- White-box monitoring: Instrumenting code from the inside (How much time does this subroutine take?)
- Every service should have its own metrics endpoint
- Hard API commitments within major versions
- New release candidate every six weeks

Main selling points

- Highly dynamic, built-in service discovery
- No hierarchical model, n-dimensional label set
- PromQL: for processing, graphing, alerting, and export
- Simple operation
- Highly efficient

Super easy to emit, parse & read

```
http_requests_total{env="prod",method="post",code="200"} 1027
http_requests_total{env="prod",method="post",code="400"} 3
http_requests_total{env="prod",method="post",code="500"} 12
http_requests_total{env="prod",method="get",code="200"} 20
http_requests_total{env="test",method="post",code="200"} 372
http_requests_total{env="test",method="post",code="400"} 75
```

PromQL

All partitions in my entire infrastructure with more than 100GB capacity that are not mounted on root?

```
node_filesystem_bytes_total{mountpoint!="/"} / 1e9 > 100
```

```
{device="sda1", mountpoint="/home", instance="10.0.0.1"} 118.8
{device="sda1", mountpoint="/home", instance="10.0.0.2"} 118.8
{device="sdb1", mountpoint="/data", instance="10.0.0.2"} 451.2
{device="xdvc", mountpoint="/mnt", instance="10.0.0.3"} 320.0
```

PromQL

What's the ratio of request errors across all service instances?

```
sum by(path) (rate(http_requests_total{status="500"}[5m])) /
 sum by(path) (rate(http_requests_total[5m]))

 {path="/status"} 0.0039
 {path="/"} 0.0011
 {path="/api/v1/topics/:topic"} 0.087
 {path="/api/v1/topics} 0.0342
```

Prometheus scale

- 1,000,000+ samples/second no problem on current hardware
- ~200,000 samples/second/core
- 16 bytes/sample compressed to 1.36 bytes/sample
- Reliable into the tens of millions of active series

Mimir

Mimir

- For **M**etrics
- Prometheus → Cortex → Grafana Enterprise Metrics → Mimir
- Scales to more than 1,000,000,000 Active Series
- Blazingly fast query performance
- Hard multi-tenancy, access control, and three-way replication
- Can ingest native OpenTelemetry, DataDog, Graphite, and Influx

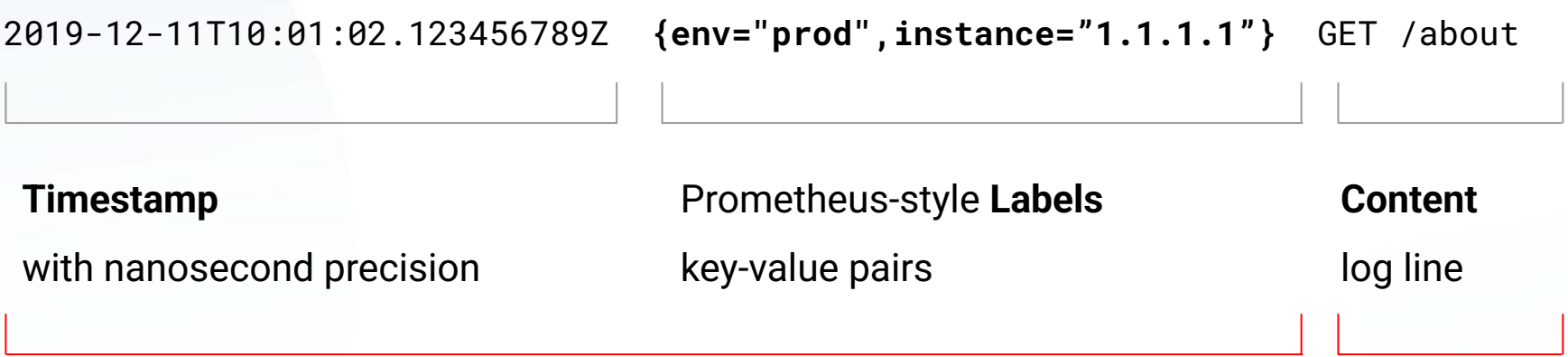
Mimir @ Grafana

- 1,000,000,000 Active Series - in one cluster
- 1,500 machines
- 7,000 CPU cores
- 30 TiB RAM

Loki

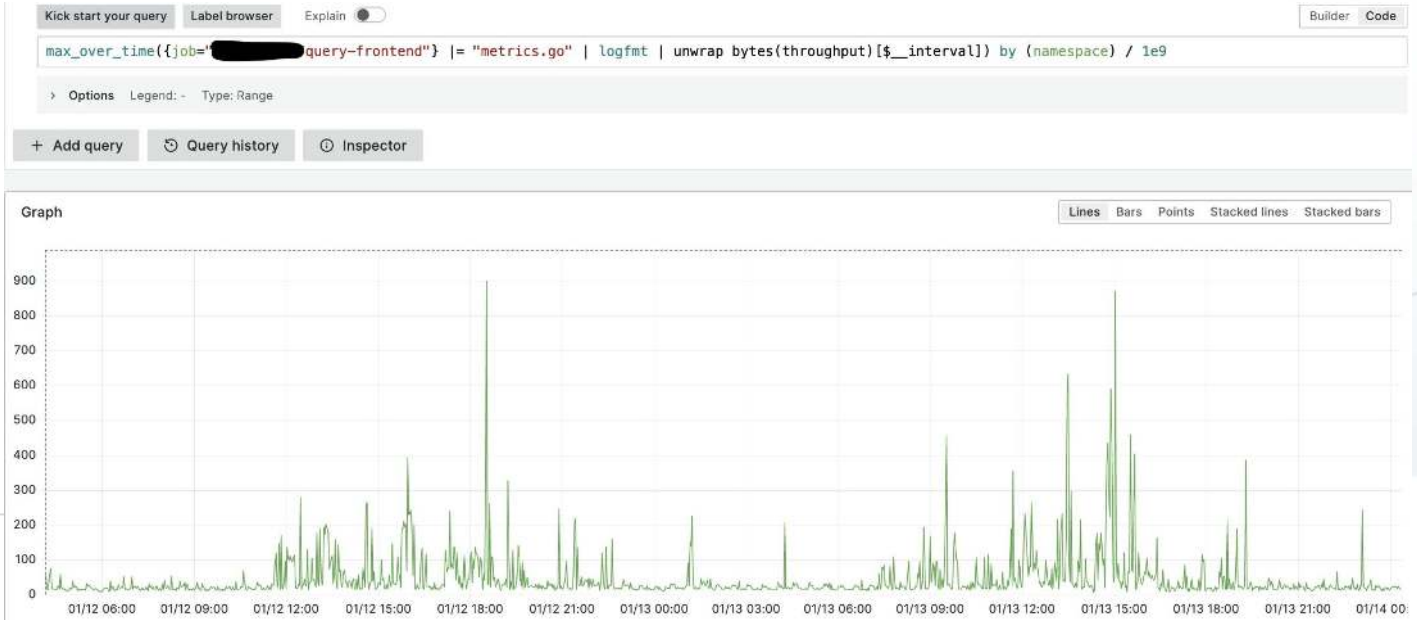
Loki 101

- For **L**ogs
- Following the same label-based system as Prometheus
  - Only index what you need often, query the rest
  - “Index the labels, query the data”
- Work with logs at scale, without the massive cost
  - Scalable low latency write path
  - Flexible schema on read
- Access logs with the same label sets as metrics
  - Turn logs into metrics, to make it easier & cheaper to work with them


2019-12-11T10:01:02.123456789Z
{env="prod",instance="1.1.1.1"}
GET /about
Timestamp
with nanosecond precision
Prometheus-style Labels
key-value pairs
Content
log line
indexed
unindexed

Loki @ Grafana Labs

- Largest user cluster: 180 TiB per day
- Queries regularly peak at **900GB/s**
  - Query 10TB in 12 seconds, including complex processing of result sets



Tempo

Tempo

- For **T**races
- Historic problem: Traces require extremely rich metadata for analysis
  - Expensive, slow, and mandates sampling
- Exemplars: Leverage the extracted logs & metrics
  - Exemplars work at Google scale, with the ease of Grafana
  - Native to Prometheus, Cortex, Thanos, and Loki
- Index and search by labelsets available for those who need it
- 100% compatible with OpenTelemetry Tracing, Zipkin, Jaeger

Tempo @ Grafana Labs

- 1,500,000 samples per second @ 450 MiB/s
  - 560 MiB/s peak
- 14-day retention @ 3 copies stored
- Latencies:
  - p99 - 2.5s
  - p90 - 2.3s
  - p50 - 1.6s

Grafana Pyroscope

Pyroscope

- For **P**rofiling
- Profiles
  - “How much CPU & RAM am I spending in what areas of the code?”
  - “...and how does this change over time?”
- Go: pprof
- Java: https://github.com/grafana/JPProf

Data (and cost) savings

Logs to metrics

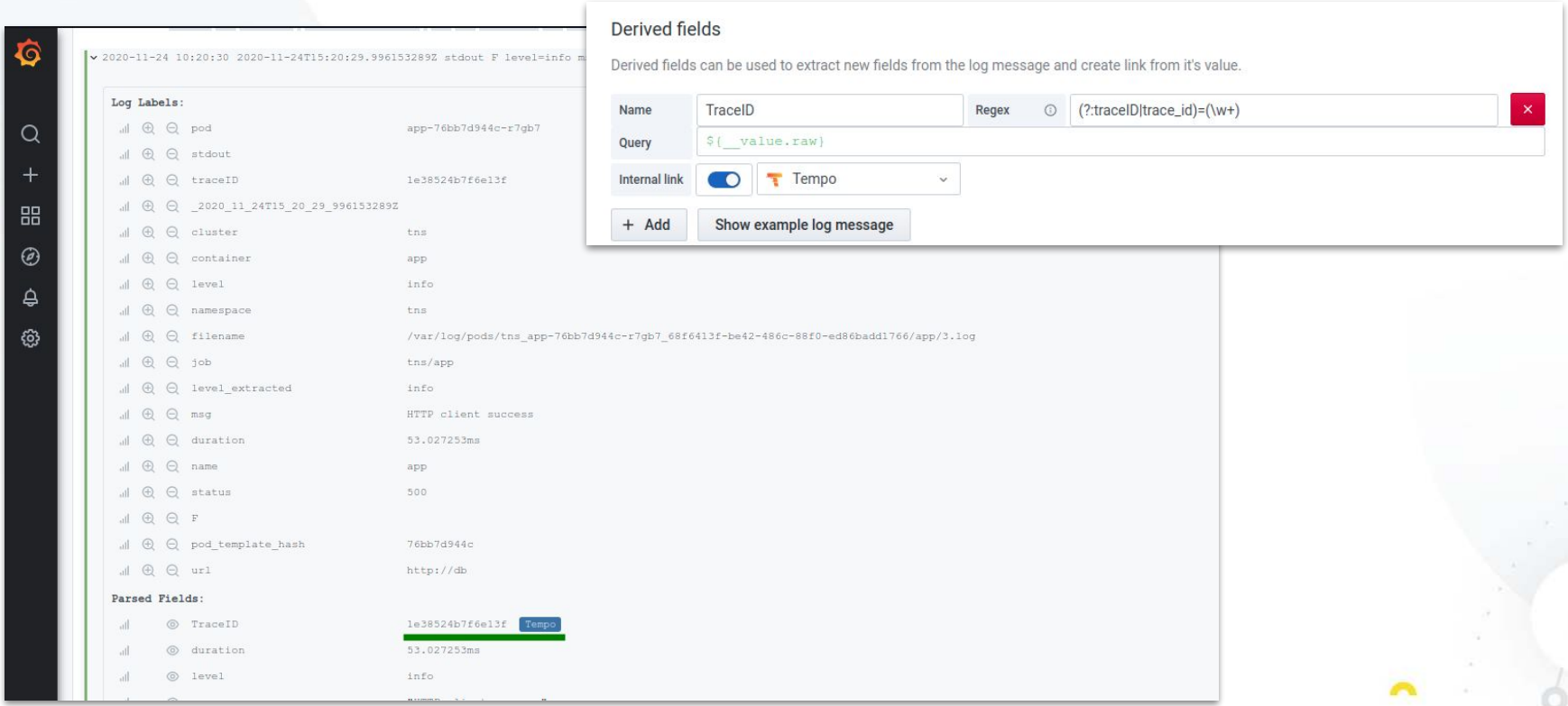
- Full text indexing: 10 TiB logs → ~20 TiB index
- Loki: 10 TiB logs → ~200 MiB index

- Logs @ Grafana ~600 Byte average per line
- Metrics ~1.36 Byte per metric sample

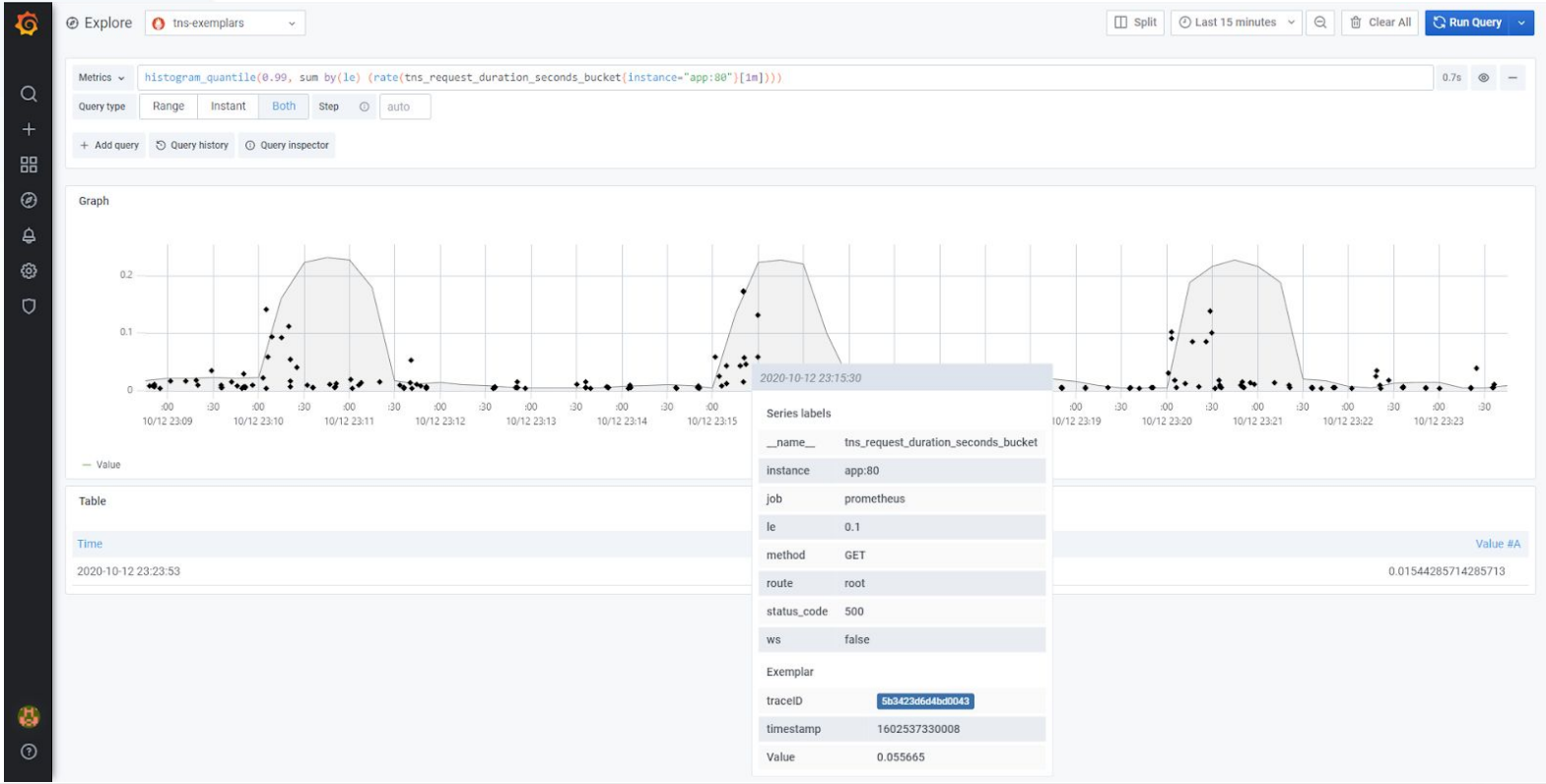
→ 99.8% reduction in storage size for first log line
~100% for every follow-up log line

Bringing it together

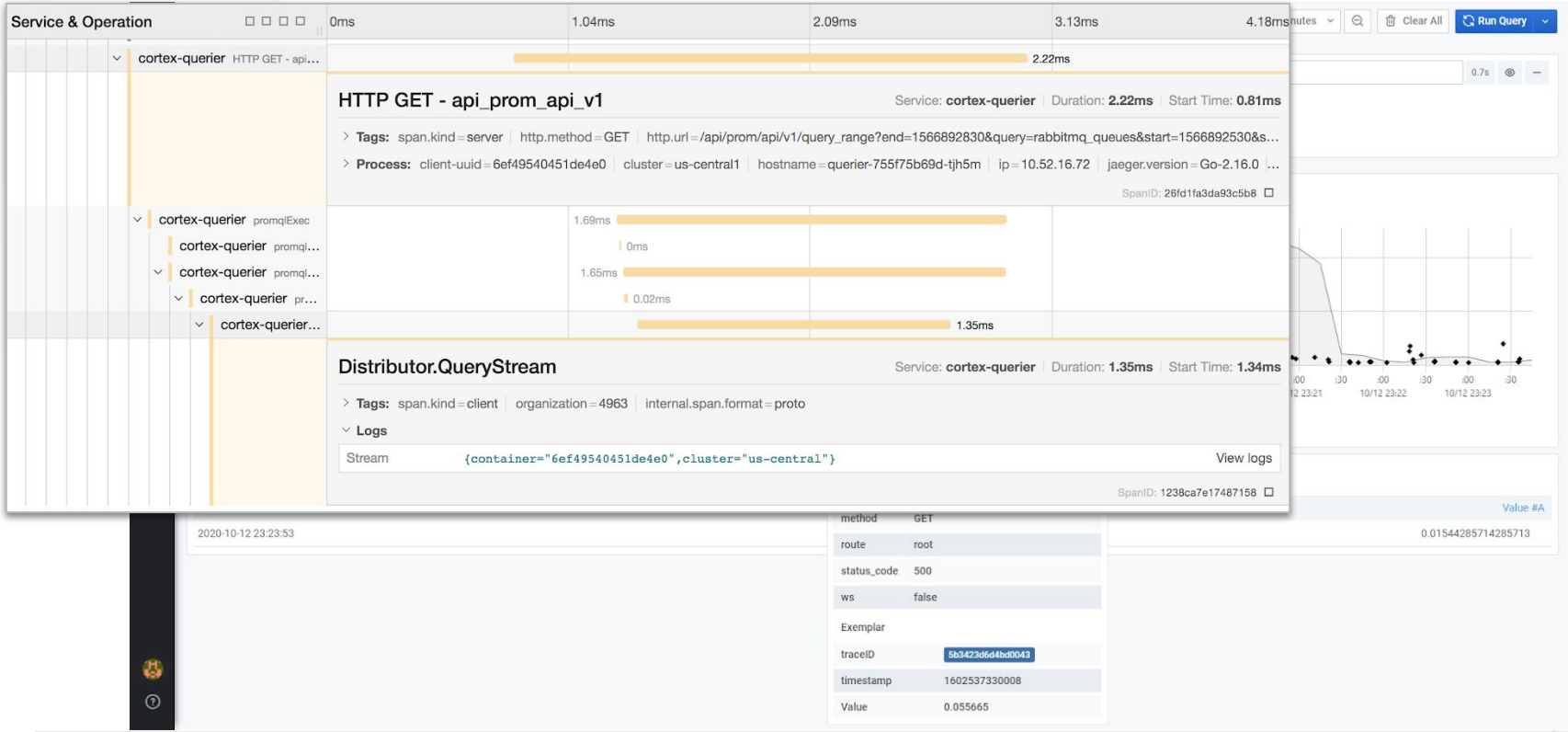
From logs to traces



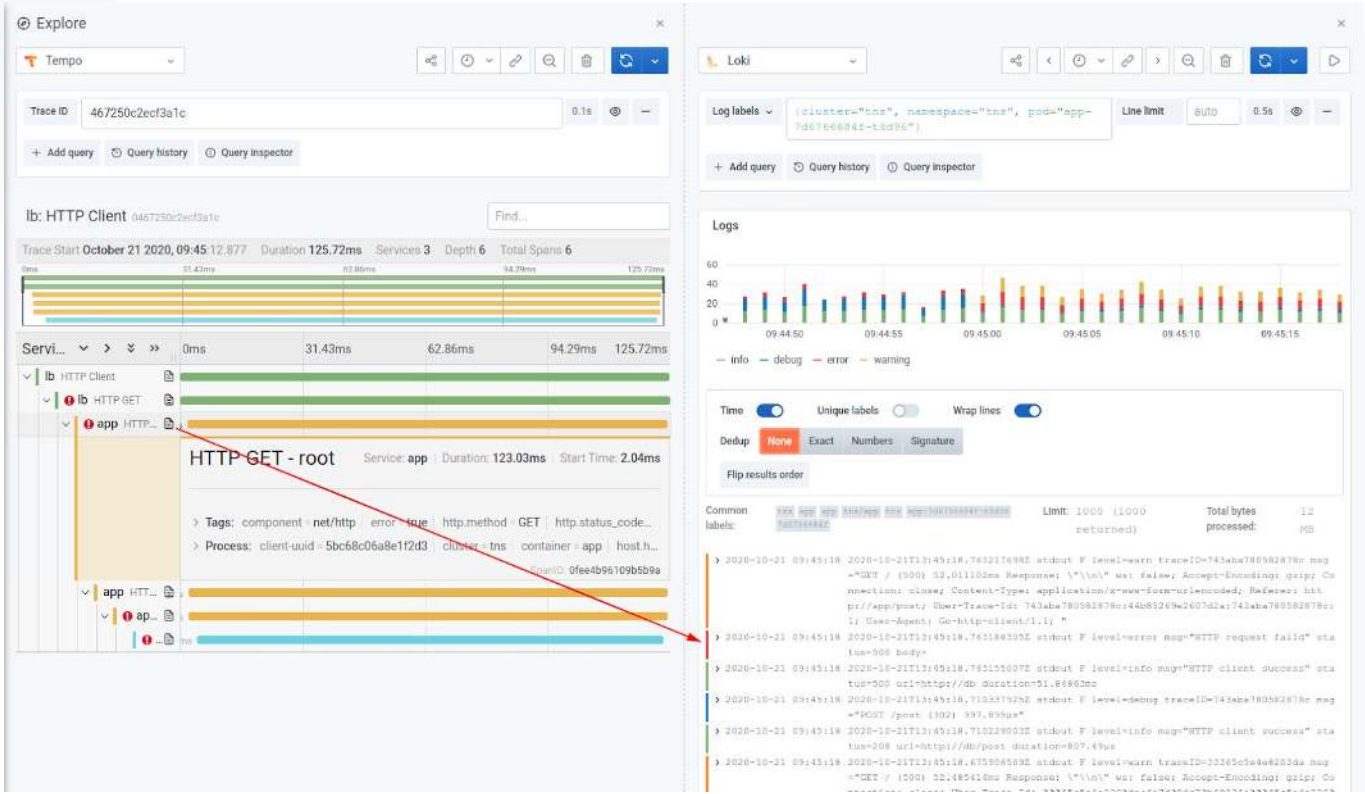
From metrics to traces

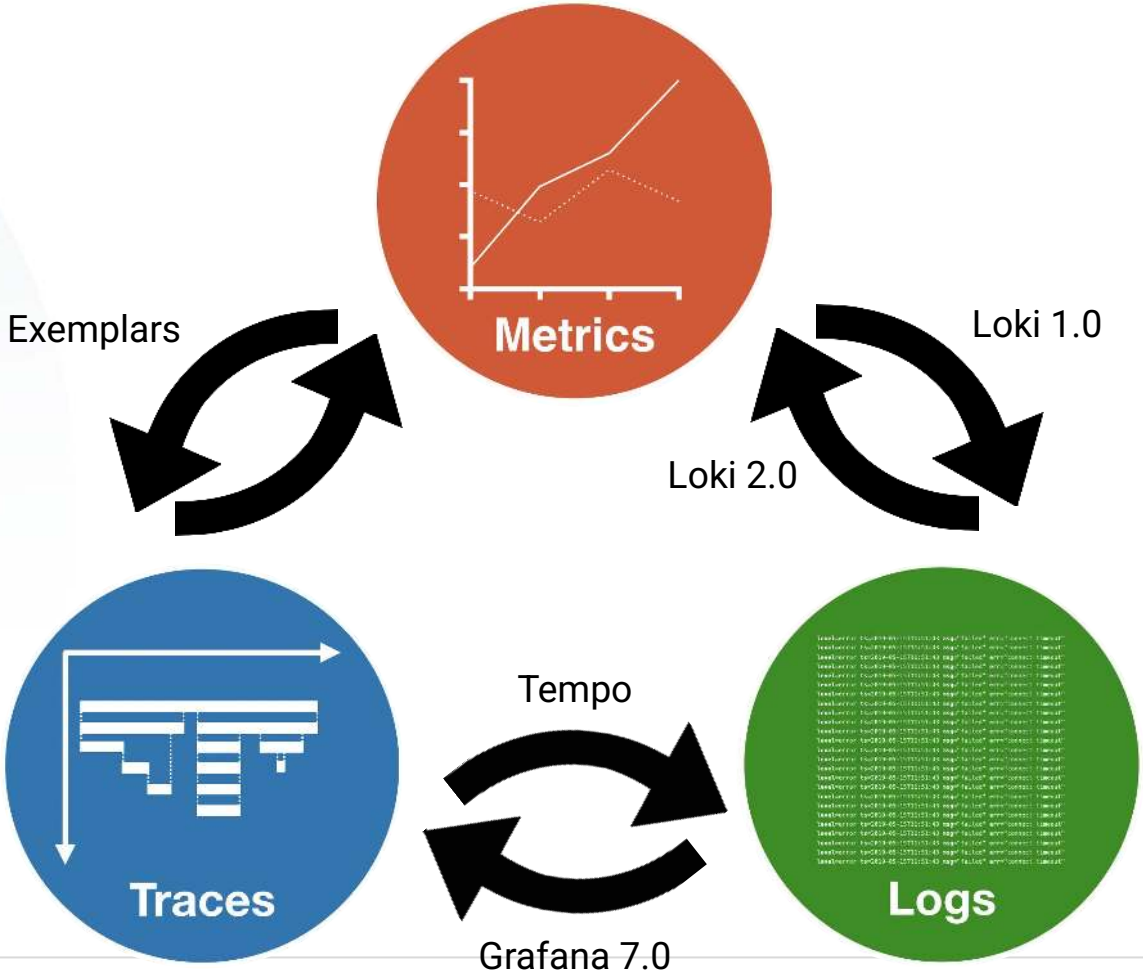


From metrics to traces

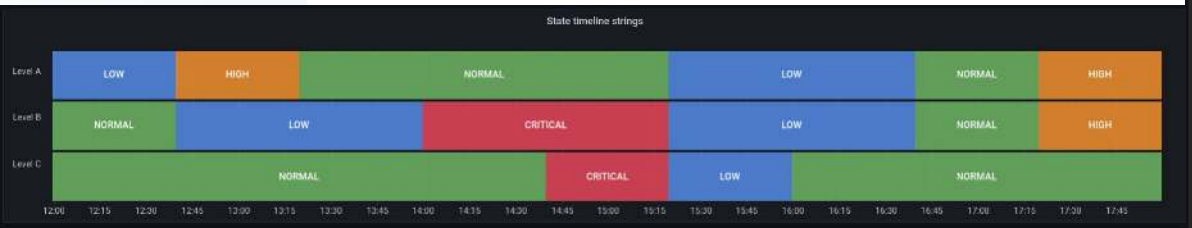


...and from traces to logs




Metrics
Exemplars
Loki 1.0
Loki 2.0
Traces
Tempo
Logs
Grafana 7.0

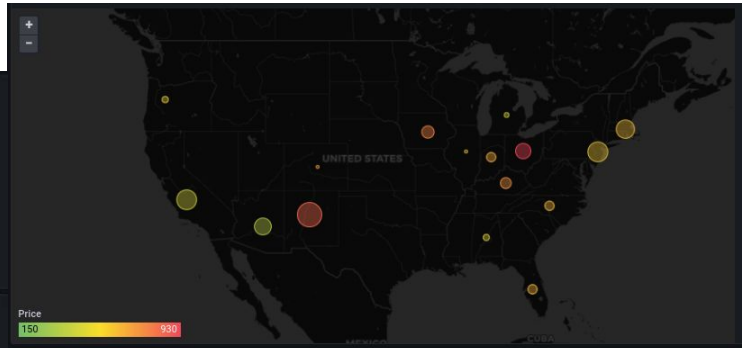
Grafana Labs


State timeline strings
Level A
Level B
Level C
LOW
HIGH
NORMAL
LOW
NORMAL
HIGH
NORMAL
LOW
CRITICAL
LOW
NORMAL
HIGH
NORMAL
CRITICAL
LOW
NORMAL
12:00 12:15 12:30 12:45 13:00 13:15 13:30 13:45 14:00 14:15 14:30 14:45 15:00 15:15 15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45

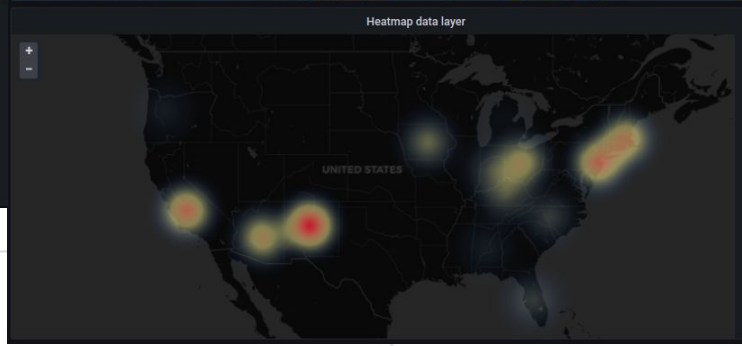

State timeline with time series + thresholds
backend_01
backend_02
backend_03
backend_04
12:00 12:15 12:30 12:45 13:00 13:15 13:30 13:45 14:00 14:15 14:30 14:45 15:00 15:15 15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45


Same query/data as above
400 °
300 °
200 °
100 °
0 °
12:00 12:15 12:30 12:45 13:00 13:15 13:30 13:45 14:00 14:15 14:30 14:45 15:00 15:15 15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45
backend_01
backend_02
backend_03
backend_04


Status history with time series + thresholds
Cellar
Living room
Porch
Bedroom
10.4 16.8 10.9 18.3 25.2 33.6 27.1 15.7 25.3 18.4 7.68 7.63
10.1 15.2 9.79 14.2 7.64 5.68 -1.0 4.76 2.69 7.69 3.26 -2.0
11.6 6.29 6.92 12.8 4.74 12.4 4.02 0.252 3.99 12.6 4.45 15.1
14.6 11.2 8.55 10.8 15.2 20.1 24.6 25.4 21.4 29.2 36.7 34.3
11:59 12:29 12:59 13:29 13:59 14:29 14:59 15:29 15:59 16:29 16:59 17:29


UNITED STATES
Price
150
930


Base layer ArcGIS wold imagery


Heatmap data layer
UNITED STATES

All of this is Open Source and you can run it yourself

(But we will also sell it to you happily)

Join us on 2nd April!

Grafana Cloud Day

Co-organized with AWS & AskMe Solutions

Date: Wednesday, April 2, 2025

Time: 12:00pm – 4:00pm Bangkok Time
(Buffet lunch is provided from 12:00pm - 1:00pm)

Venue: Mövenpick BDMS Connect Center, Pathum Wan

Register: bit.ly/gcday2025

Thank you!

richih@richih.org
chaos.social/@RichiH
github.com/RichiH/talks