



# That's not a lot of data

Leveraging Cloud-Native Technologies for Automotive



Richard "RichiH" Hartmann



Hugh Blemings

My name is Hugh Blemings and I am an Engineering Director at Grafana Labs in our Databases Department

First our apologies -

- My colleague Richi Hartman who prepared this presentation was called away at short notice on a family matter
- Richi sends his apologies for being unable to attend
- I will attempt to do his presentation justice!

I will provide an overview of Cloud Native Observability and some thoughts on how it can be applied to Automotive

There will be time left for questions at the end



# How humanity deals with data

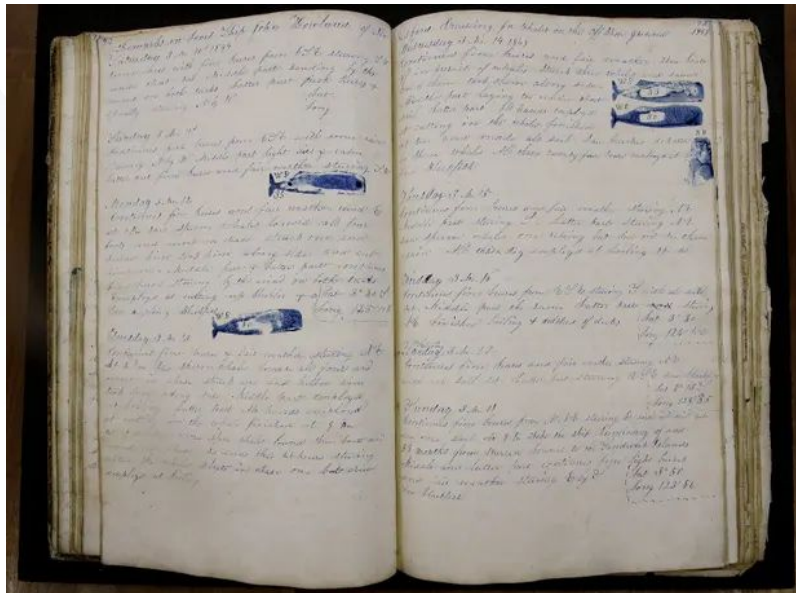
Humanity has always optimized how it deals with data



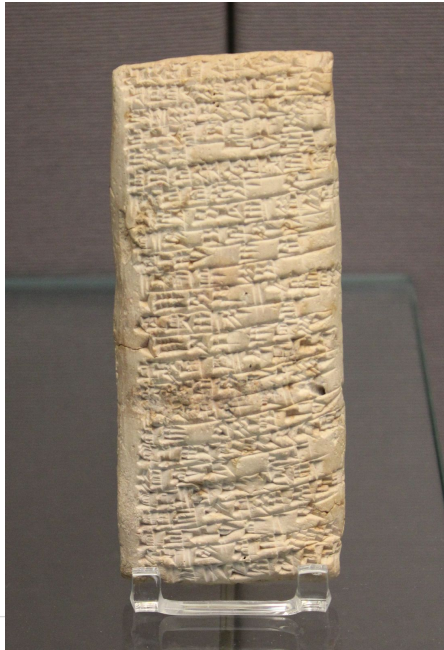


This is the Stahlschlüssel, a reference text used in Germany which contains the properties of steel, copper and virtually all materials available world wide at the time

- again lots and lots of important numbers



- Going further back, Mariners kept track of weather conditions and whales that they sighted and caught
- These texts have renewed purpose today as scientists work through them and the historically important numerical weather and other data they contain



- The first “letter” we know of is 4000 years old
- It’s about the wrong grade of copper being delivered, provisional payment still being made
  - and the seller being rude to the buyer’s representative.



- Going back even further, this is the oldest writing we currently know of
- 6000 years old
- Most likely a record of who owned what at the time

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



Again and again and again






# Observability & SRE

Or: Buzzwords, and their useful parts



*[...] 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**

- In fact, we've used the term observability for hundreds of years
- Gauss and Kálmán should need no introduction to this audience!

# Observability, the buzzword

- Observability is a superset of Monitoring
  - 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

# 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

Hugh Skip

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

- A mainframe application and a microservices fleet are fundamentally the same
- Microservices broke up old service and system boundaries
  - Enabling horizontal scalability, arguably at the cost of vertical scalability

- One can argue that these are new terms for familiar concepts
- You don't need fancy just something that works

# SRE, an instantiation of DevOps

- 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 primary

SRE - Those who write the code maintain the code

# Observability helps shared understanding

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

## Observability helps shared understanding

- Devs know about flow of information
- Ops about operational constraints
- Product Managers about product direction
- owners about customer pains



# Alerting

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

**Anything currently or imminently impacting  
a customer Service must be alerted upon**

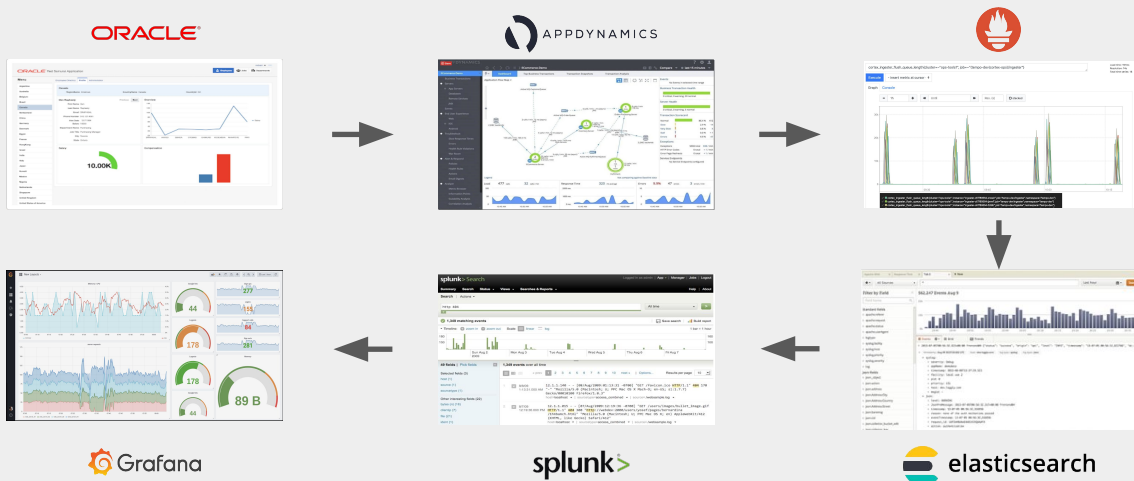
**But nothing(!) else**

## Alerting

- Don't just measure/record it - alert on it!
- Be selective - noisy alerting burns your people out
- Having better tooling will generate better alerts

So what is the reality today of all these tools and components ?

# Today's reality of monitoring and observability: Disparate systems. Disparate data.



## Reality of monitoring today

- Costs you time, money and nerves
- You can lose context switching between systems
- Naming conventions differ so hard to automate
- Teams like to make their own decisions too

So lets talk about some tools that work well together and to our initial point, fit well into the human desire to optimise how we use our data



# Prometheus

# Prometheus 101

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

# What is 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)
    - Or of MAF sensor :)
  - Service latency (histograms)

# Data format 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
```

Prometheus uses a very simple data format

- Can be implemented trivially in C with a printf and a few more lines of code
- Easy to integrate with embedded systems

# 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
  - Suitable for real time telemetry
- ~200,000 samples/second/core
- 16 bytes/sample compressed to 1.36 bytes/sample
- Reliable into the tens of millions of active series

## Prometheus

- Suitable for real time telemetry
- In use by F1 teams for this very purpose

Next I will talk a little bit about some of Grafana's Observability tools  
All are available as Open Source, Commercial and Cloud based offerings



# Mimir

# Mimir

- For **M**etrics
- Prometheus → Cortex → Grafana Enterprise Metrics → Mimir
- Scales to more than 1,000,000,000 Active Series
- Blazingly fast ingest and query performance
  - Fast enough for realtime data
- Hard multi-tenancy, access control and highly reliable
- 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 Logs
- 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

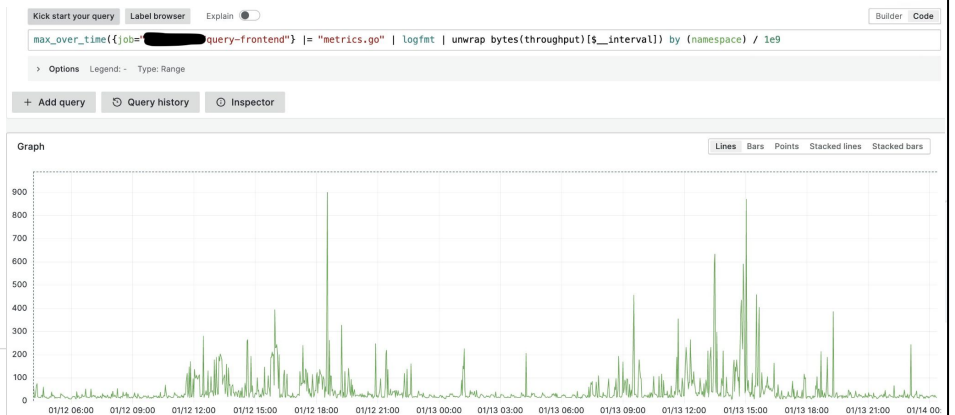
unindexed

Here we have a sample log line.

- It has 3 elements
  - Timestamp
  - Labels
  - Content
- Loki only indexes the timestamp and the labels.

# 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 Traces
- Historic problem: Traces require extremely rich metadata for analysis
  - Expensive, slow, and mandates sampling
- Exemplars: Leverage the extracted logs & metrics
  - Allow Traces to be treated as metrics or logs
  - 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

Exemplars are extracted from the traces into log and metric data

# 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**rofil**i**ng
- Profiles
  - "How much CPU & RAM am I spending in what areas of the code?"
    - "...and how does this change over time?"

# Data (and cost) savings

# Converting 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



All Open Source  
and/or  
<http://play.grafana.com>



# How does this apply to automotive?

# Automotive - I

- Several market leaders already use these technologies
- It doesn't matter if 100,000 Kubernetes pods send data, or 100,000 cars
- Efficient
  - 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

# Automotive - II

- Tools are fast enough for telemetry at least up to tens of kHz sample rates
  - Used in motorsport at all levels
- Geospatial data can be ingested as metrics and visualised
- Leading AI/ML companies use Grafana tools to monitor their training environments
  - Important to get the most out of those expensive GPUs



All of this is Open Source and you can run it yourself  
(But we will also happily sell it to you)



Richi and I are both Engineers at heart,  
So we're not really here to sell things to you!  
But we do see many companies using Grafana cloud so we take care of their  
observability infrastructure for them which lets their teams focus on their business  
needs

# Q&A



 Grafana Labs



# Thank you!

[richih@richih.org](mailto:richih@richih.org) / [hugh@blemings.org](mailto:hugh@blemings.org)  
[github.com/RichiH/talks](https://github.com/RichiH/talks)



 Grafana Labs

