Sascha Rauch
ATIX AG
rauch@atix.de
@insomnia_io
insomnia_io

Vincent Welker
ATIX AG
welker@atix.de

Keeping the Overview in Chaos
Monitoring & Tracing in Kubernetes

Hi, I'm Sascha

Hi, I'm Vincent

Requesttracing
Learning to understand your traffic

Do you even trace?

Tracing is a method for IT and DevOps teams to monitor applications

Especially those composed of microservices

Not exclusive for microservices

Finding errors in requests gets harder the more services are involved

More complex infrastructure = More fun with tracing

Our goal is to completely understand how our systems behave (and why)

This is also known by companies planning their projects

Therefore monitoring and logging is usually part of the initial planning

Monitoring improves our understanding of how systems behave

Logging helps to track error reports and associated data centrally

Where tracing?

Tracing is the logical consequence of monitoring and logging

Pinpoint where failures occur and what causes suboptimal performance

Powerful tool for understanding what happens in our systems

Tracing often fails to get attention in the initial planning of systems

All three components are the basis for analysis, operation and optimization

Without tracing there is no real observability

If distributed tracing is so valuable, why doesn’t everyone do it already?

Distributed tracing is challenging

Or is it?

Instrumentation must propagate the tracing context both within and between processes

Accomplishing this touches almost every part of an application

It is not reasonable to ask everyone to use a single tracing vendor

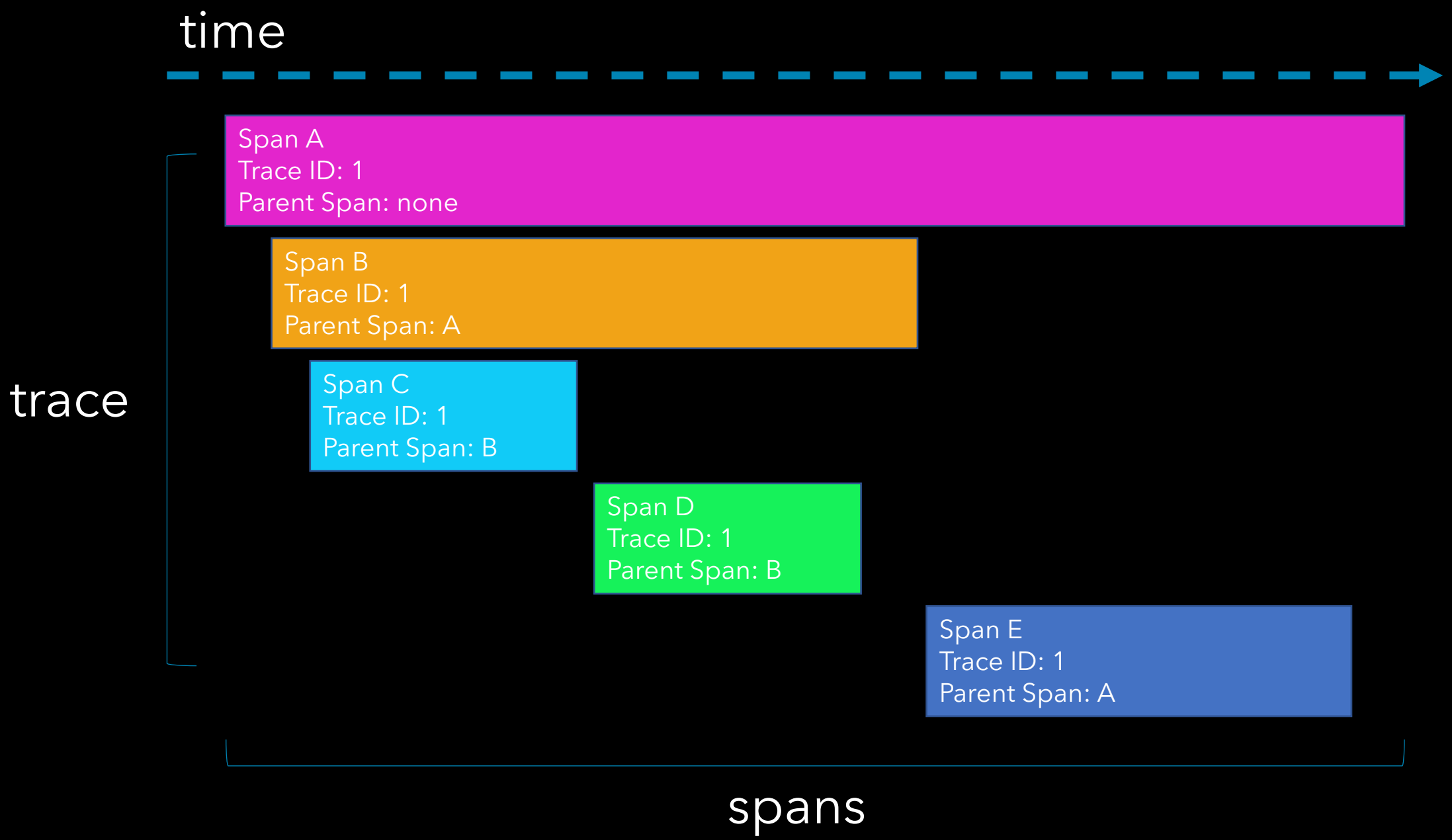
We’re not here for problems

OpenTelemetry standard allows developers to instrument their code

Independent of the programming language

Easy integration to standard components

This goes without binding to any particular tracing vendor


time
Span A
Trace ID: 1
Parent Span: none
Span B
Trace ID: 1
Parent Span: A
trace
Span C
Trace ID: 1
Parent Span: B
Span D
Trace ID: 1
Parent Span: B
Span E
Trace ID: 1
Parent Span: A
spans

Every component of a distributed system is instrumented in isolation

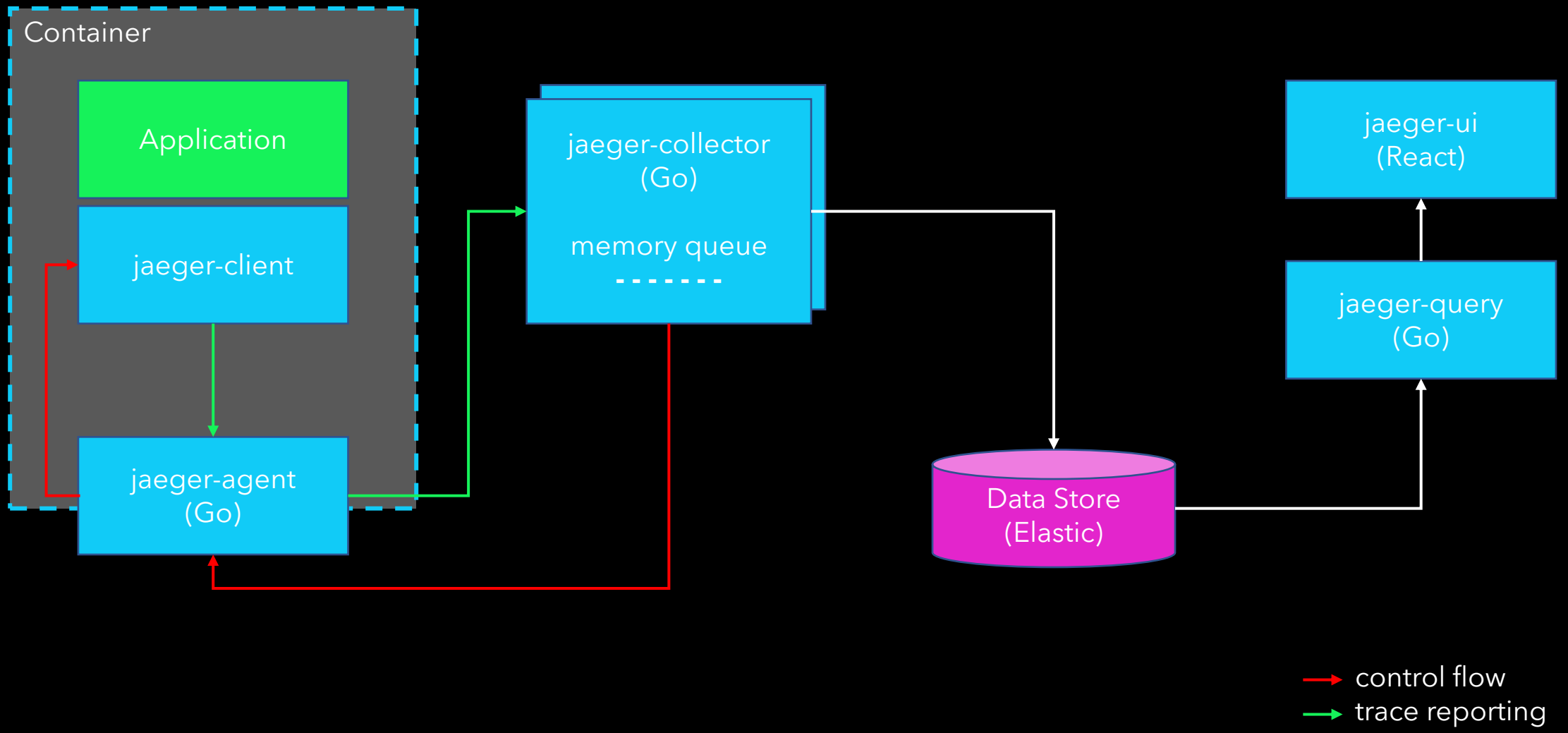
We can choose a downstream tracing technology per configuration change

Introducing Jaeger

OpenTelemetry compatible data model and instrumentation libraries

Jaeger integrates well in kubernetes, lightweight backends written in go

Supports various storage backends such as Elastic, Cassandra and Kafka


Container
Application
jaeger-client
jaeger-agent
(Go)
jaeger-collector
(Go)
memory queue
jaeger-ui
(React)
jaeger-query
(Go)
Data Store
(Elastic)
control flow
trace reporting

In conclusion

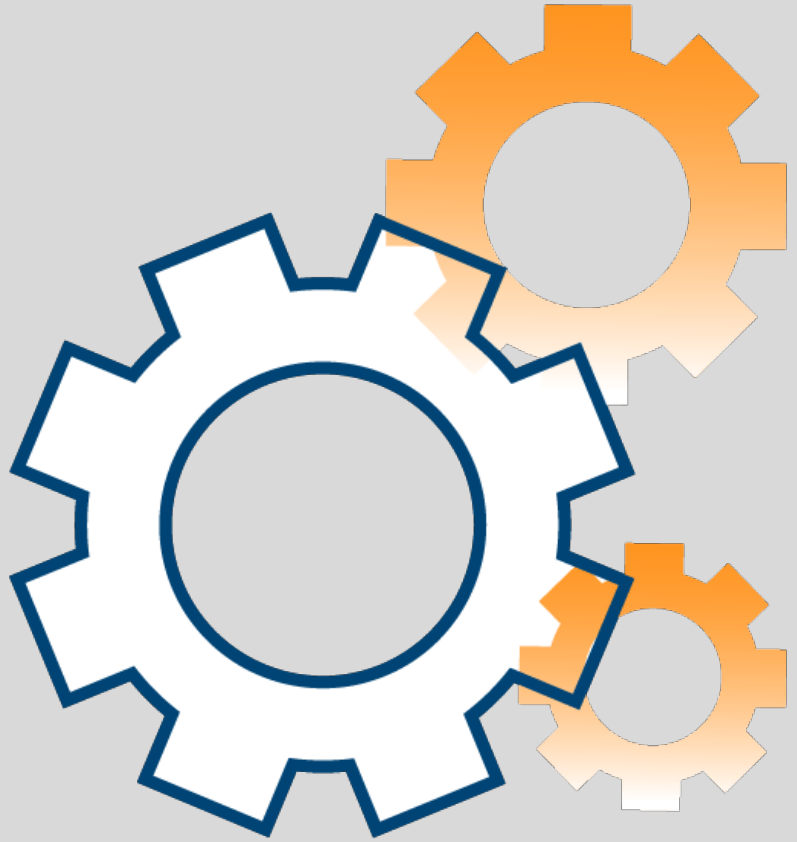
Jaeger provides an easy integration of Open Tracing in kubernetes

While logging and monitoring are mostly set up in clusters, tracing is missing most of the time

Without tracing you will always have a blind spot (which is bad)

ATIX

The Linux & Open Source Company



Monitoring

with Prometheus

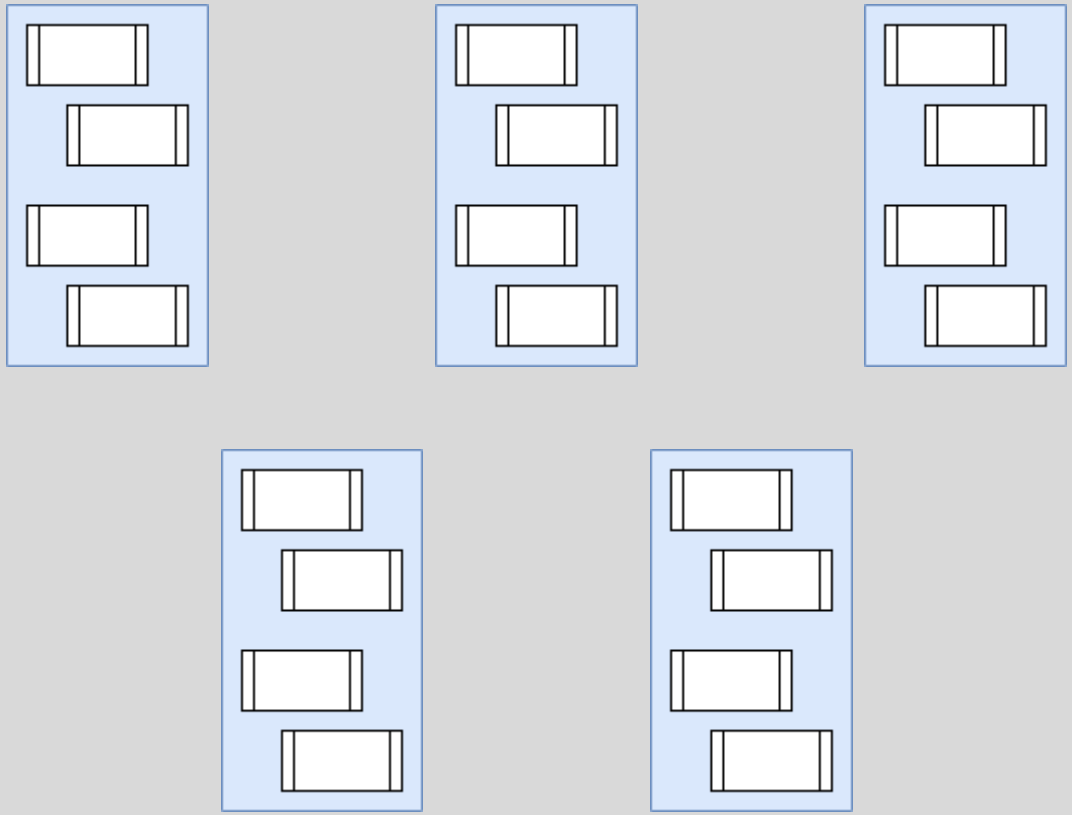
What is Prometheus

- Open source project founded by SoundCloud in 2012

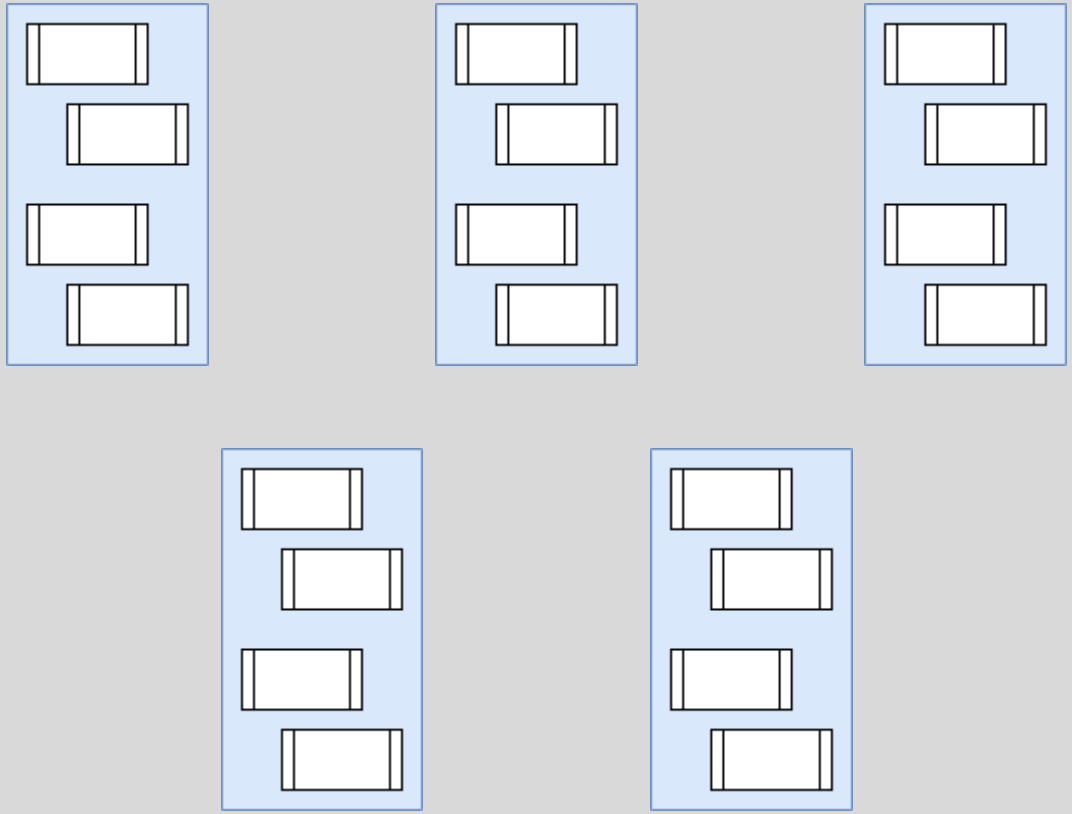
- Monitoring tool

- Works very well in dynamic container environments

Why use Prometheus?

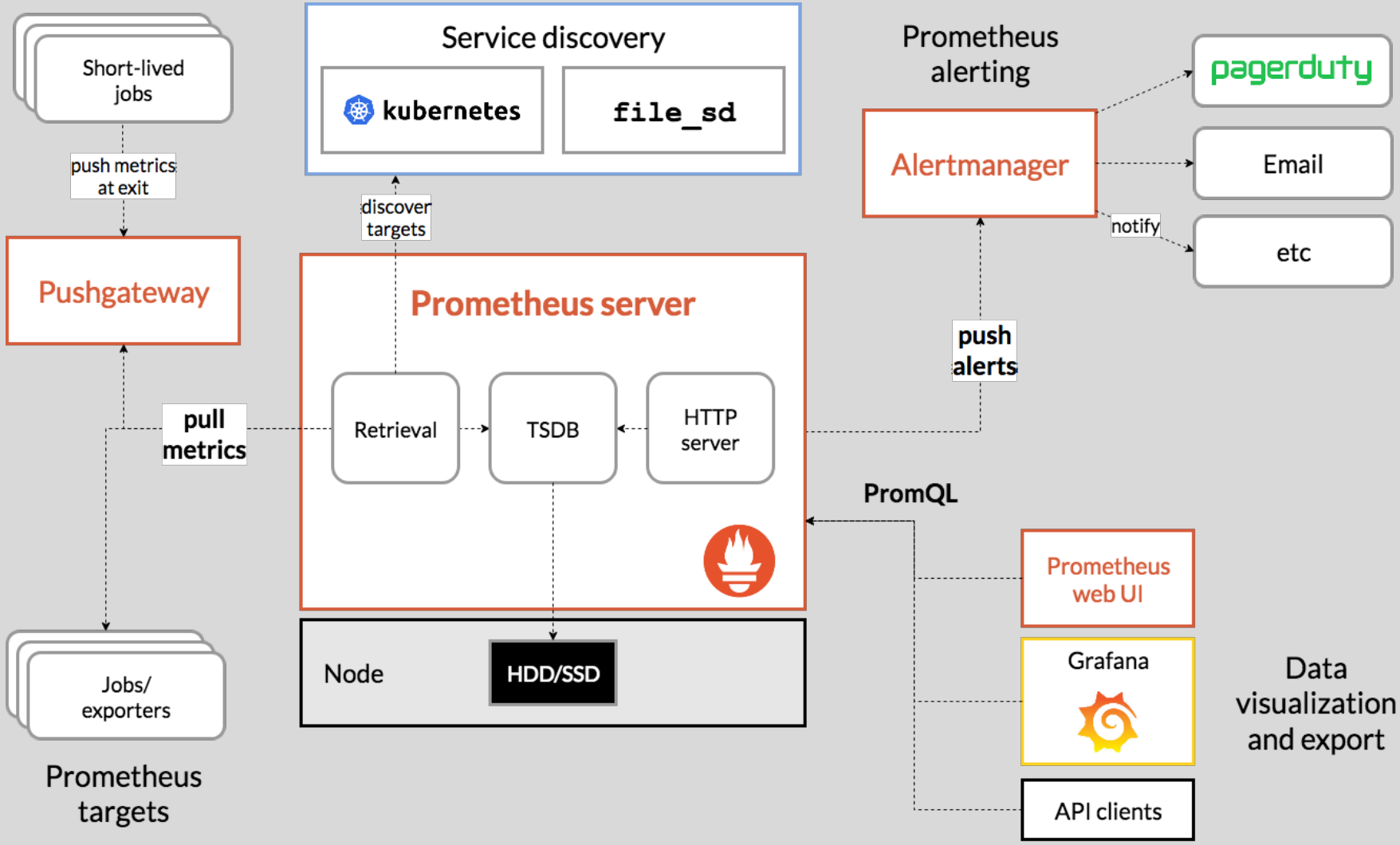


Why use Prometheus?



W

Architecture

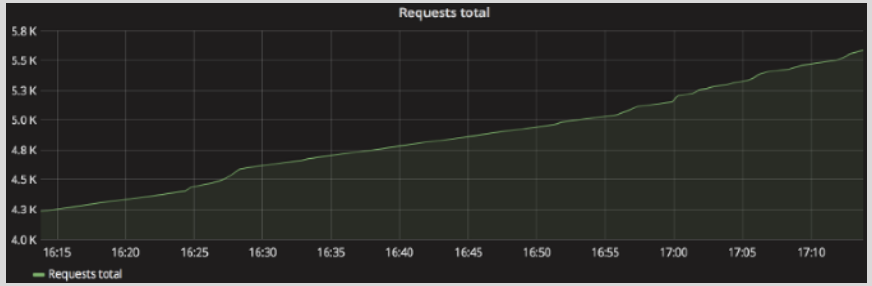


Alertmanager

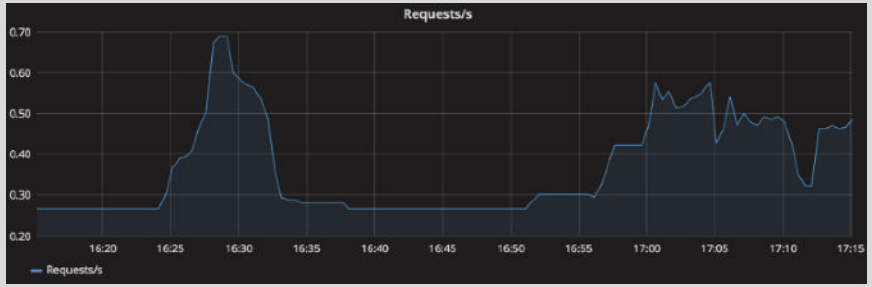
- Deduplicating, grouping, and routing of alerts to the correct receiver
- Integration such as email, PagerDuty, or OpsGenie.
- Also takes care of silencing and inhibition of alerts.

Metrics types

- Counter
  - A *counter* is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart.



- Gauge
  - A *gauge* is a metric that represents a single numerical value that can arbitrarily go up and down.



Metrics types

- Histogram
  - A *histogram* samples observations (usually
    response sizes) and counts them in configu
    of all observed values.
    - cumulative counters for the observation bu
    - the **total sum** of all observed values
    - the **count** of events that have been observe

```
# HELP caller_duration_calls The duration of processed calls
# TYPE caller_duration_calls histogram
caller_duration_calls_bucket{le="0"} 0
caller_duration_calls_bucket{le="1"} 0
caller_duration_calls_bucket{le="2"} 0
caller_duration_calls_bucket{le="4"} 0
caller_duration_calls_bucket{le="6"} 0
caller_duration_calls_bucket{le="10"} 0
caller_duration_calls_bucket{le="16"} 0
caller_duration_calls_bucket{le="26"} 0
caller_duration_calls_bucket{le="+Inf"} 0
caller_duration_calls_sum 0
caller_duration_calls_count 0
```

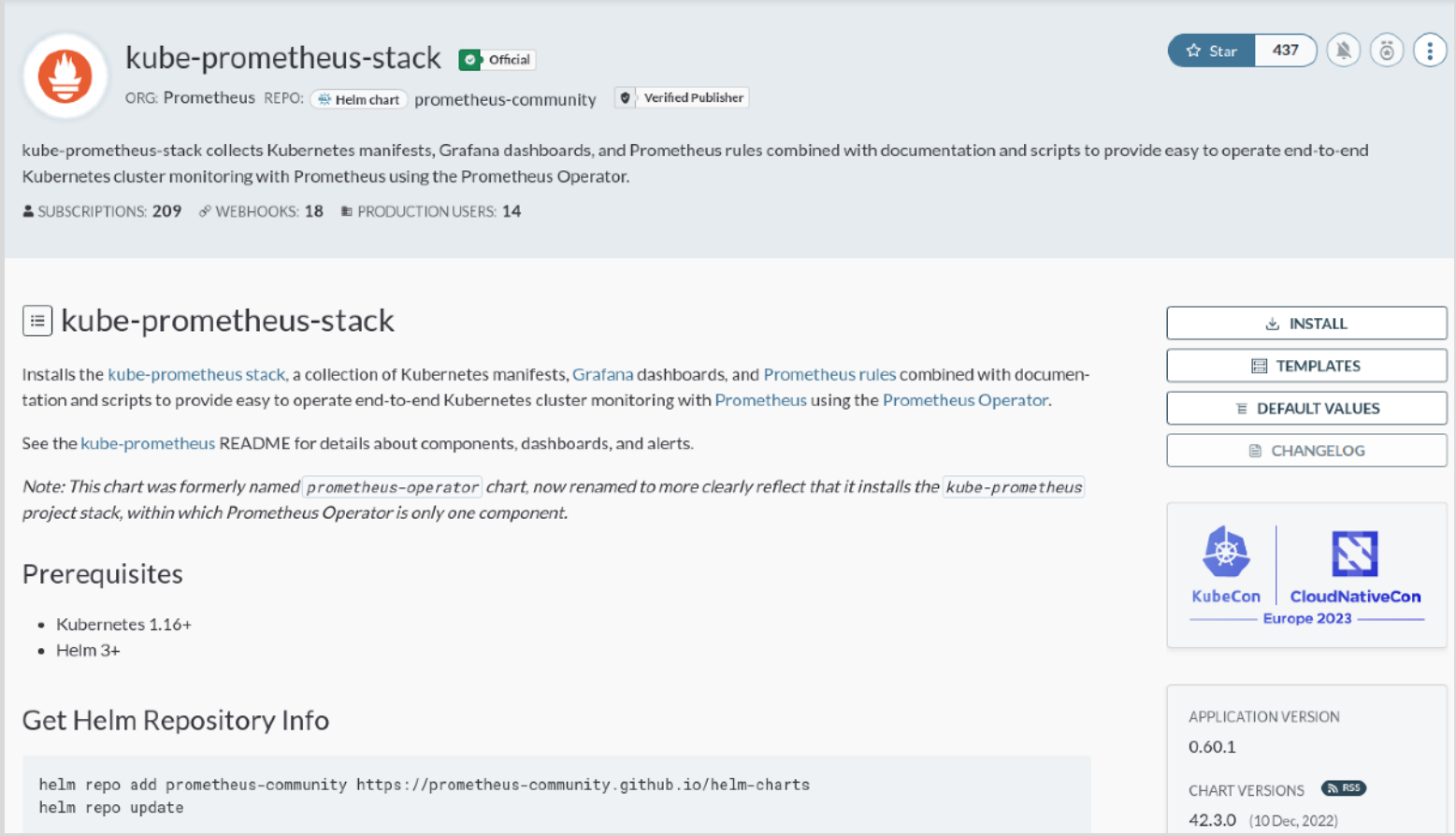
- Summary
  - Histograms and Summa
    expose the distribution
    - Histograms use samplin
      Prometheus server
    - Summaries are calculate

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 1.8296e-05
go_gc_duration_seconds{quantile="0.25"} 6.4095e-05
go_gc_duration_seconds{quantile="0.5"} 9.4908e-05
go_gc_duration_seconds{quantile="0.75"} 0.000114411
go_gc_duration_seconds{quantile="1"} 0.000147806
go_gc_duration_seconds_sum 0.005659768
go_gc_duration_seconds_count 64
```

Metrics types – Bottom Line

- **Counters**: use for counting events that happen (e.g. total number of requests) and query using rate()
- **Gauge**: use to instrument the current state of a metric (e.g. memory usage, jobs in queue)
- **Histograms**: use to sample observations in order to analyse distribution of a data set (e.g. request latency)
- **Summaries**: use for pre-calculated quantiles on client side, but be mindful of calculation cost and aggregation limitations

Prometheus on a K8s-Infrastructure



kube-prometheus-stack Official

ORG: Prometheus REPO: Helm chart prometheus-community Verified Publisher

Star 437

kube-prometheus-stack collects Kubernetes manifests, Grafana dashboards, and Prometheus rules combined with documentation and scripts to provide easy to operate end-to-end Kubernetes cluster monitoring with Prometheus using the Prometheus Operator.

SUBSCRIPTIONS: 209 WEBHOOKS: 18 PRODUCTION USERS: 14

kube-prometheus-stack

Installs the kube-prometheus stack, a collection of Kubernetes manifests, Grafana dashboards, and Prometheus rules combined with documentation and scripts to provide easy to operate end-to-end Kubernetes cluster monitoring with Prometheus using the Prometheus Operator.

See the kube-prometheus README for details about components, dashboards, and alerts.

Note: This chart was formerly named `prometheus-operator` chart, now renamed to more clearly reflect that it installs the `kube-prometheus` project stack, within which Prometheus Operator is only one component.

Prerequisites

- Kubernetes 1.16+
- Helm 3+

Get Helm Repository Info

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
```

INSTALL

TEMPLATES

DEFAULT VALUES

CHANGELOG

KubeCon CloudNativeCon Europe 2023

APPLICATION VERSION

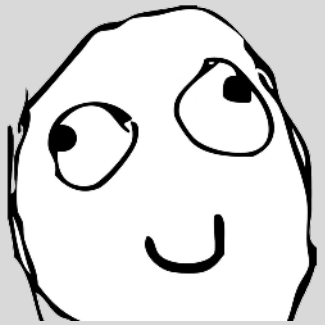
0.60.1

CHART VERSIONS RSS

42.3.0 (10 Dec, 2022)

https://artifacthub.io/packages/helm/prometheus-community/kube-prometheus-stack

But I don't use Kubernetes



- Deploy a node exporter on a VM or Linux Server

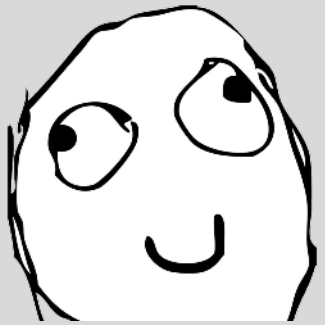
```
wget
https://github.com/prometheus/node_exporter/releases/download/v1.3.1/
node_exporter-1.3.1.linux-amd64.tar.gz

tar xvfz node_exporter-1.3.1.linux-amd64.tar.gz

./node_exporter
```

```
curl http://localhost:9100/metrics
```

But I don't use Kubernetes



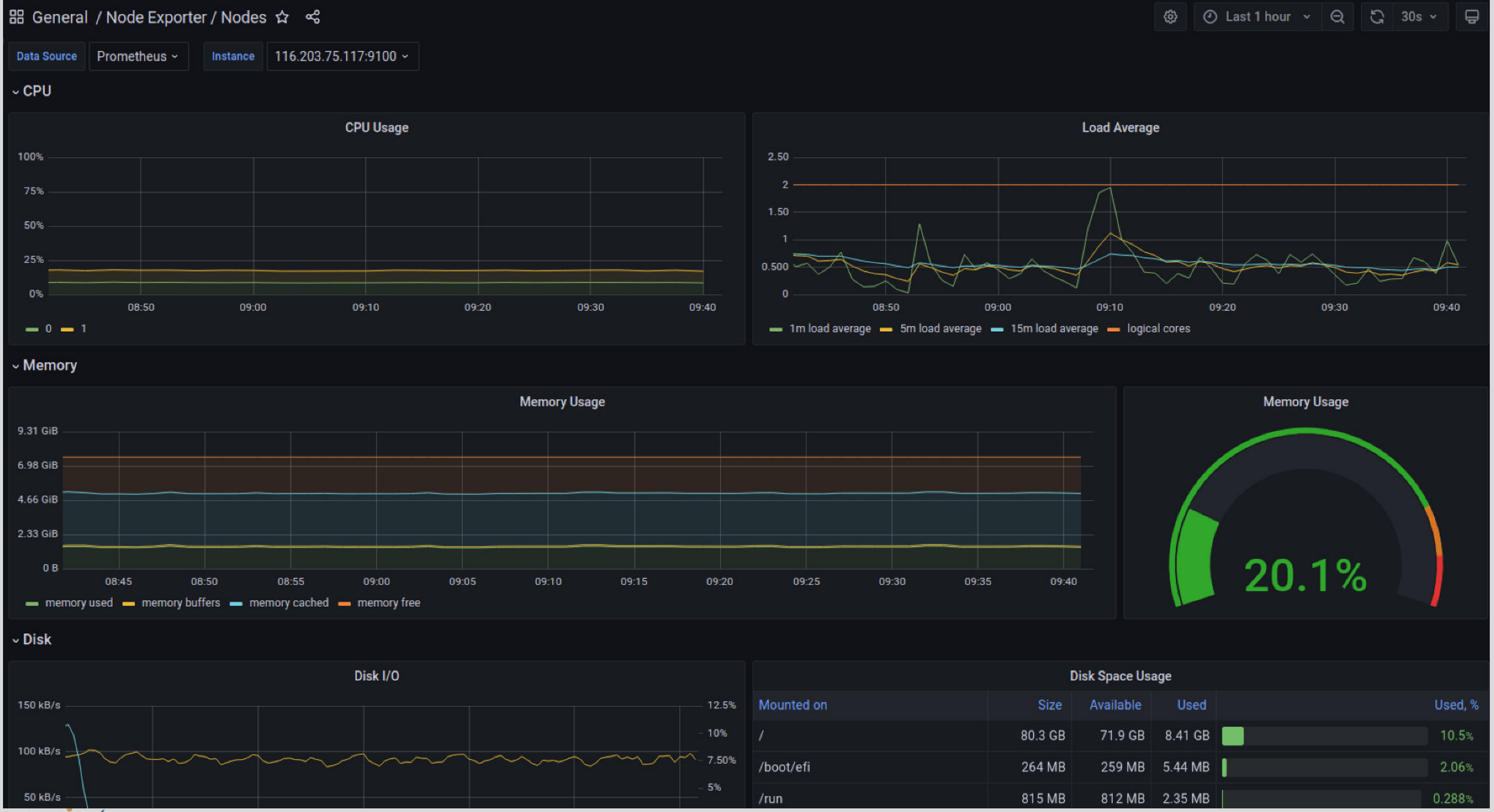
- Deploy a prometheus grafana stack with docker compose

```yaml
services:
  prometheus:
    image: prom/prometheus:v2.34.0
  container_name: prometheus
  networks:
    - prometheus-nw
  ports:
    - "9090:9090"
  volumes:
    - ${PWD}/prometheus.yml:/etc/prometheus/prometheus.yml
grafana:
  image: grafana/grafana:8.4.4-ubuntu
  container_name: grafana
  networks:
    - prometheus-nw
  ports:
    - "3000:3000"
```

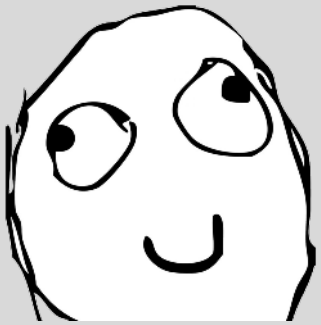
```yaml
global:
  scrape_interval: 10s
scrape_configs:
  - job_name: my-little-linux-vm
    metrics_path: /metrics
    static_configs:
      - targets: [ 'localhost:9100' ]
```

What units do we monitor?

- CPU Status
- Memory Usage
- Disk Space Usage
- Request Count
- Request Duration
- Exceptions Count

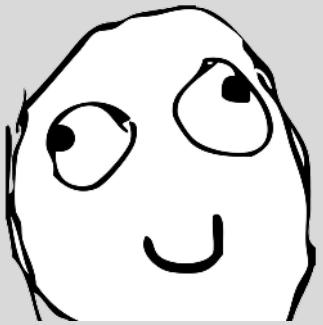

General / Node Exporter / Nodes
Last 1 hour
30s
Data Source
Prometheus
Instance
116.203.75.117:9100
CPU
CPU Usage
100%
75%
50%
25%
0%
08:50
09:00
09:10
09:20
09:30
09:40
0
1
Load Average
2.50
2
1.50
1
0.500
0
1m load average
5m load average
15m load average
logical cores
Memory
Memory Usage
9.31 GiB
6.98 GiB
4.66 GiB
2.33 GiB
0 B
08:45
08:50
08:55
09:00
09:05
09:10
09:15
09:20
09:25
09:30
09:35
09:40
memory used
memory buffers
memory cached
memory free
Memory Usage
20.1%
Disk
Disk I/O
150 kB/s
100 kB/s
50 kB/s
12.5%
10%
7.50%
5%
Disk Space Usage
Mounted on
Size
Available
Used
Used, %
/
80.3 GB
71.9 GB
8.41 GB
10.5%
/boot/efi
264 MB
259 MB
5.44 MB
2.06%
/run
815 MB
812 MB
2.35 MB
0.288%

But I want to monitor my application

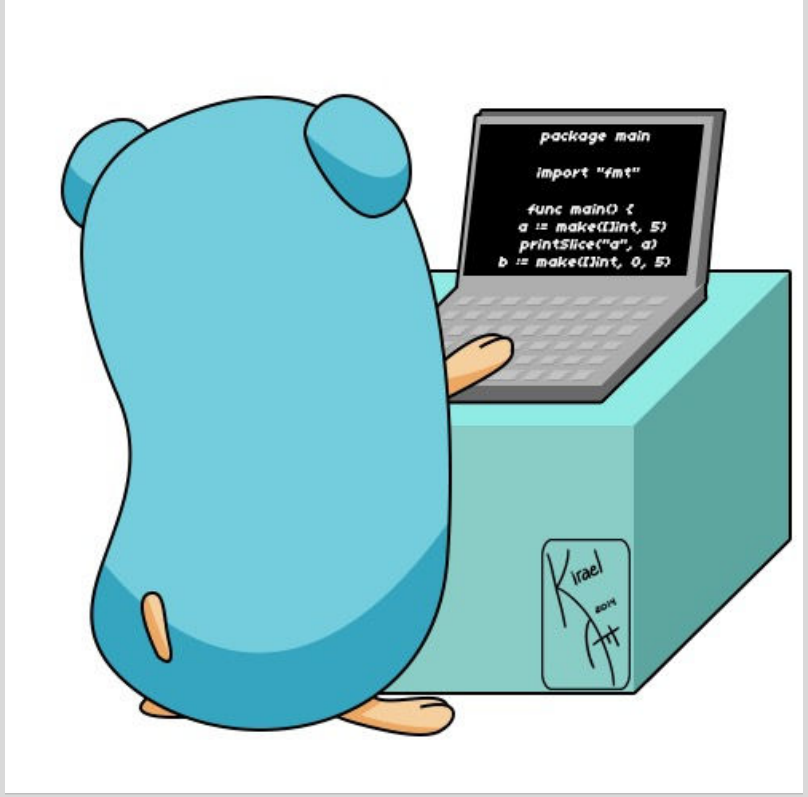


- Prometheus provides client libraries for:
  - Go
  - Java
  - Python
  - Ruby
  - .Net

But I want to monitor my application



- Prometheus provides client libraries for:
  - Go
  - Java
  - Python
  - Ruby
  - .Net



```go
var (
    clientCallsDuration = promauto.NewHistogram(prometheus.HistogramOpts{
        Name:    "caller_duration_calls",
        Help:    "The duration of processed calls",
        Buckets: []float64{0, 1, 2, 4, 6, 10, 16, 26},
    })
)

func LoadRecordResponse(succesful bool, durationSeconds float64) {
    go func() {
        clientCallsDuration.Observe(durationSeconds)
    }()
}

func InitAsync() {
    go func() {
        Init()
    }()
}

func Init() {
    http.Handle("/metrics", promhttp.Handler())
    http.ListenAndServe(":2112", nil)
}

```

Code Changes

- Start the exporter endpoint (on a concurrent thread)

```
73      monitoring.InitAsync()
74      http.Handle("/", NewHandler(http.HandlerFunc(handleReq), "handleRequest"))
75      http.HandleFunc("/healthz", handleHealthCheck)
76      http.ListenAndServe(fmt.Sprintf(":%v", Configuration.Port), nil)
```

- Measure the client-call duration and call the defined function

```
144     start := time.Now()
145     response, err := client.Do(req)
146     elapsedSeconds := time.Since(start).Seconds()
```

```
176         response.Body.Close()
177         monitoring.LoadRecordResponse(true, elapsedSeconds)
```

/metrics

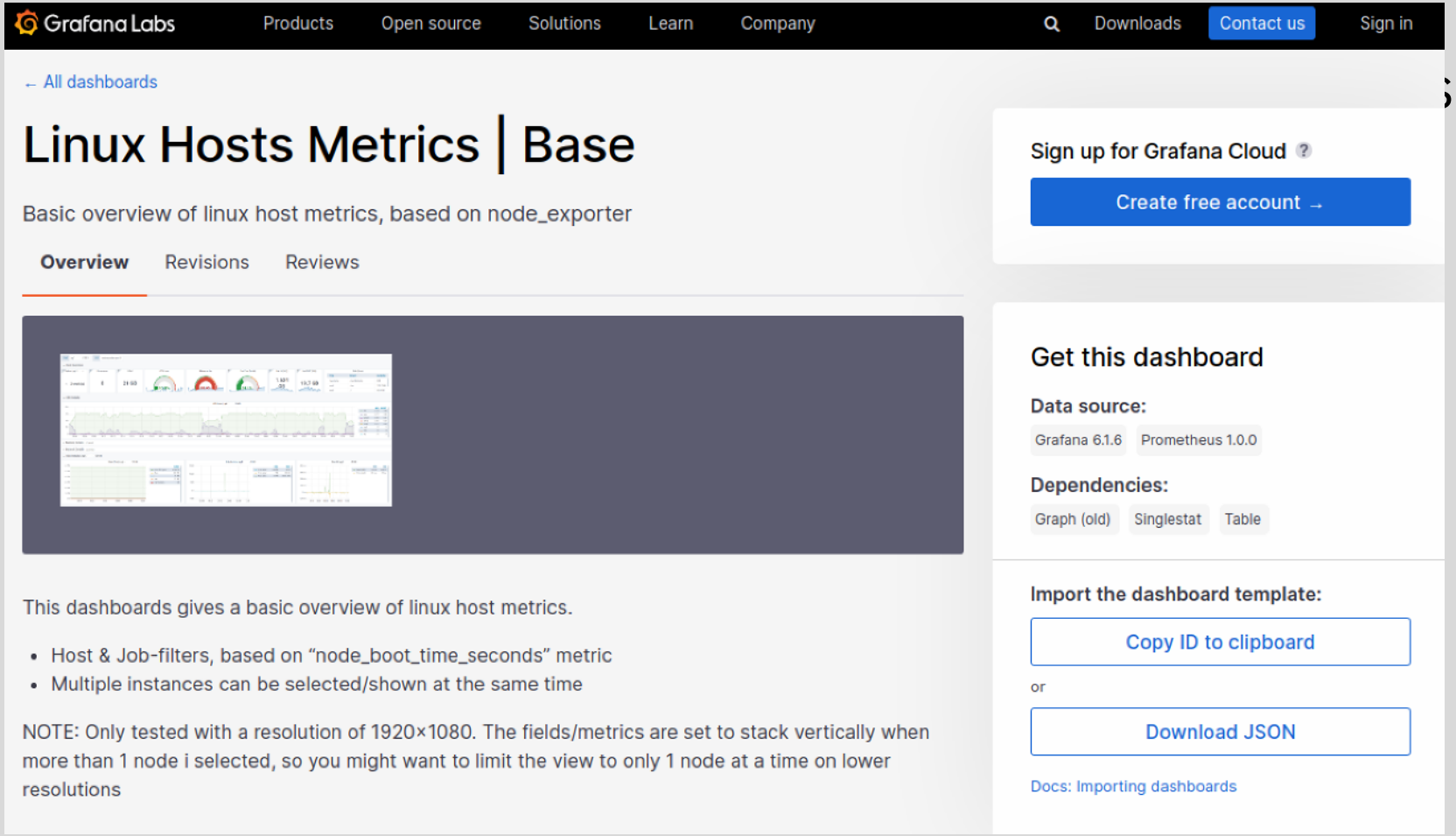
```
# HELP caller_duration_calls The duration of processed calls
# TYPE caller_duration_calls histogram
caller_duration_calls_bucket{le="0"} 0
caller_duration_calls_bucket{le="1"} 0
caller_duration_calls_bucket{le="2"} 0
caller_duration_calls_bucket{le="4"} 0
caller_duration_calls_bucket{le="6"} 3
caller_duration_calls_bucket{le="10"} 4
caller_duration_calls_bucket{le="16"} 4
caller_duration_calls_bucket{le="26"} 4
caller_duration_calls_bucket{le="+Inf"} 4
caller_duration_calls_sum 21.051199964
caller_duration_calls_count 4
```

But wait…

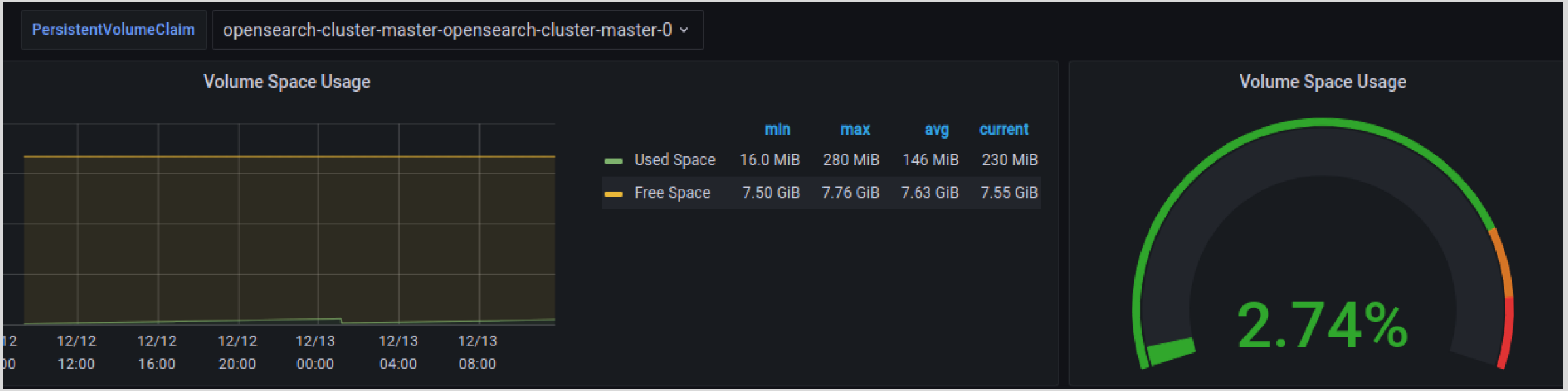
```
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.19.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.532352e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 5.45614496e+08
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 5007
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 3.205671e+06
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 9.010864e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.532352e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 4.063232e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
```

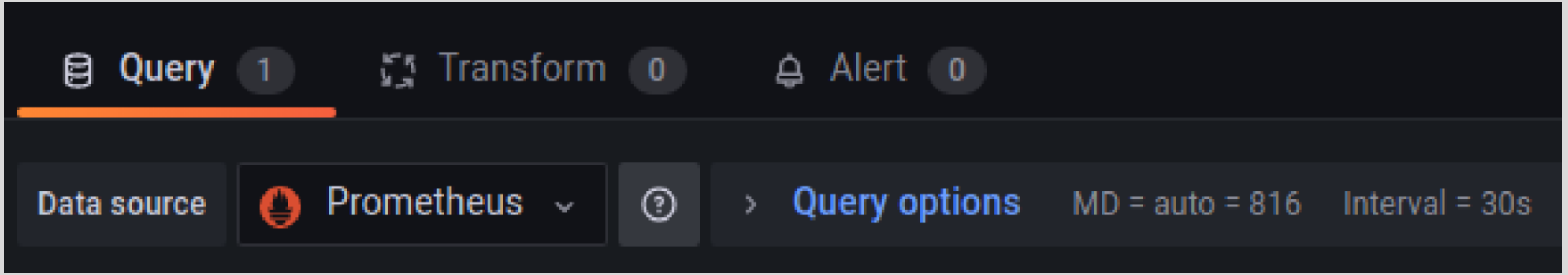
General Advice for Grafana

- spread

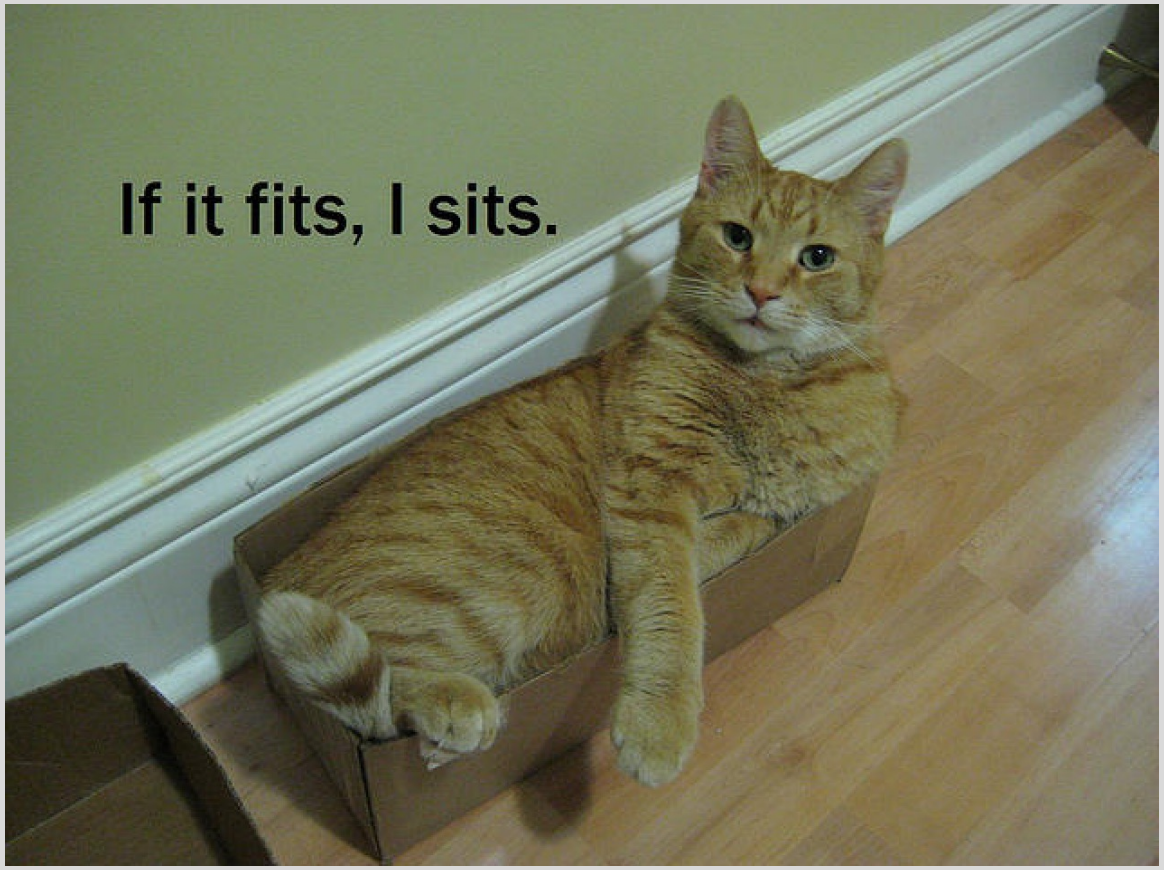


Alerting via Grafana





When does it fit?



DEMO

Linux Stammtisch
23.03. – 18:30 – remote

Slides?
atix.de/chemnitzer-linux-tage-2023

Sascha Rauch
ATIX AG
rauch@atix.de
@insomnia_io
insomnia_io

Vincent Welker
ATIX AG
welker@atix.de

Questions?