Building *Observable* Go Services

GopherCon Turkey - 2020

Who am I?

SWE/SRE **@Red Hat**

Observability Platform Team

Thanos Maintainer

Promethues Contributor

kkakkoyun

kakkoyun

What is Observability?

> *In control theory, observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs.*
>
> *- Wikipedia*

Observability 101

FOSDEM 19 .org

On Observability

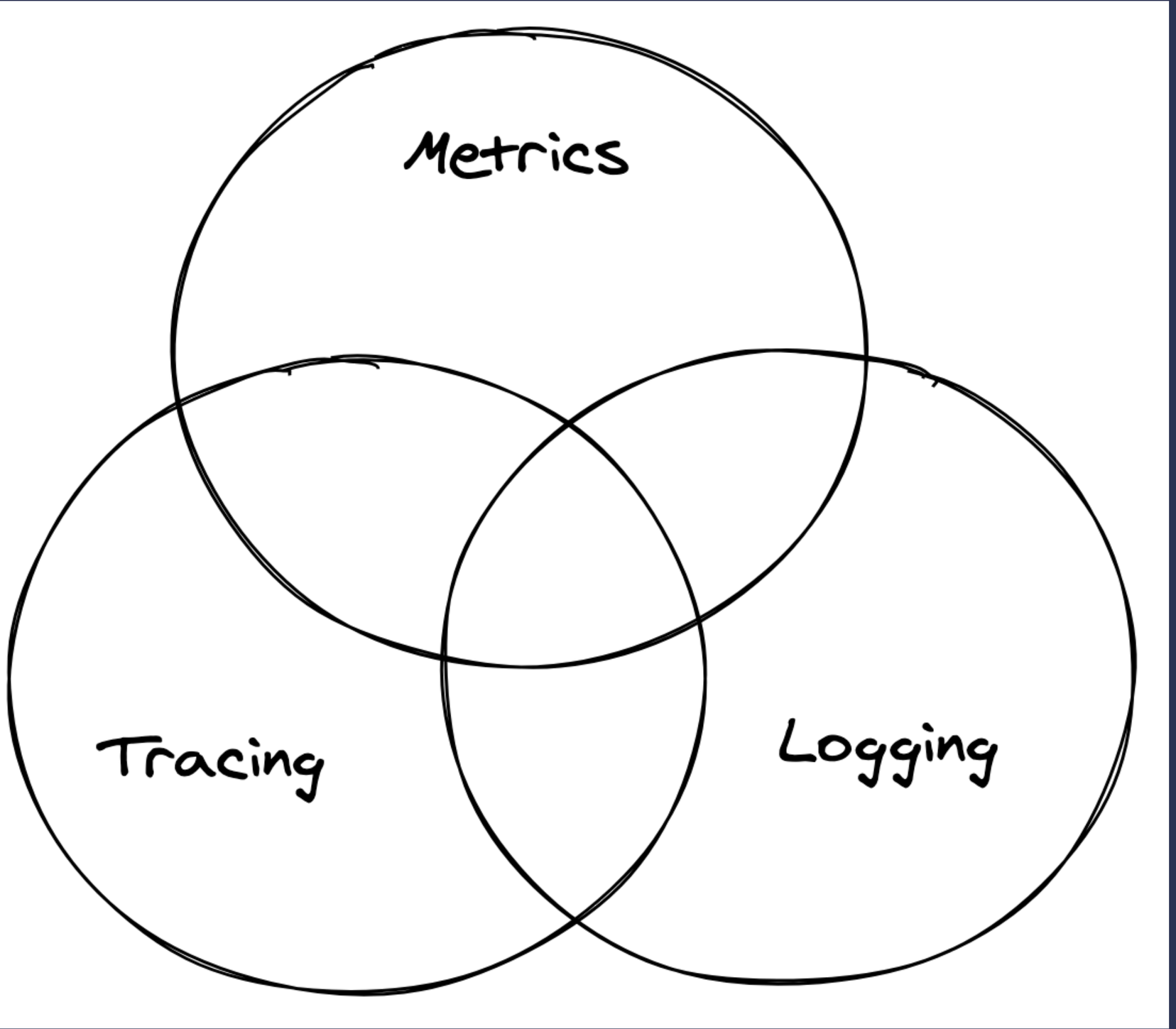
Observability 101

Richard Hartmann

UB2.252A (Lameere)
2019-02-03

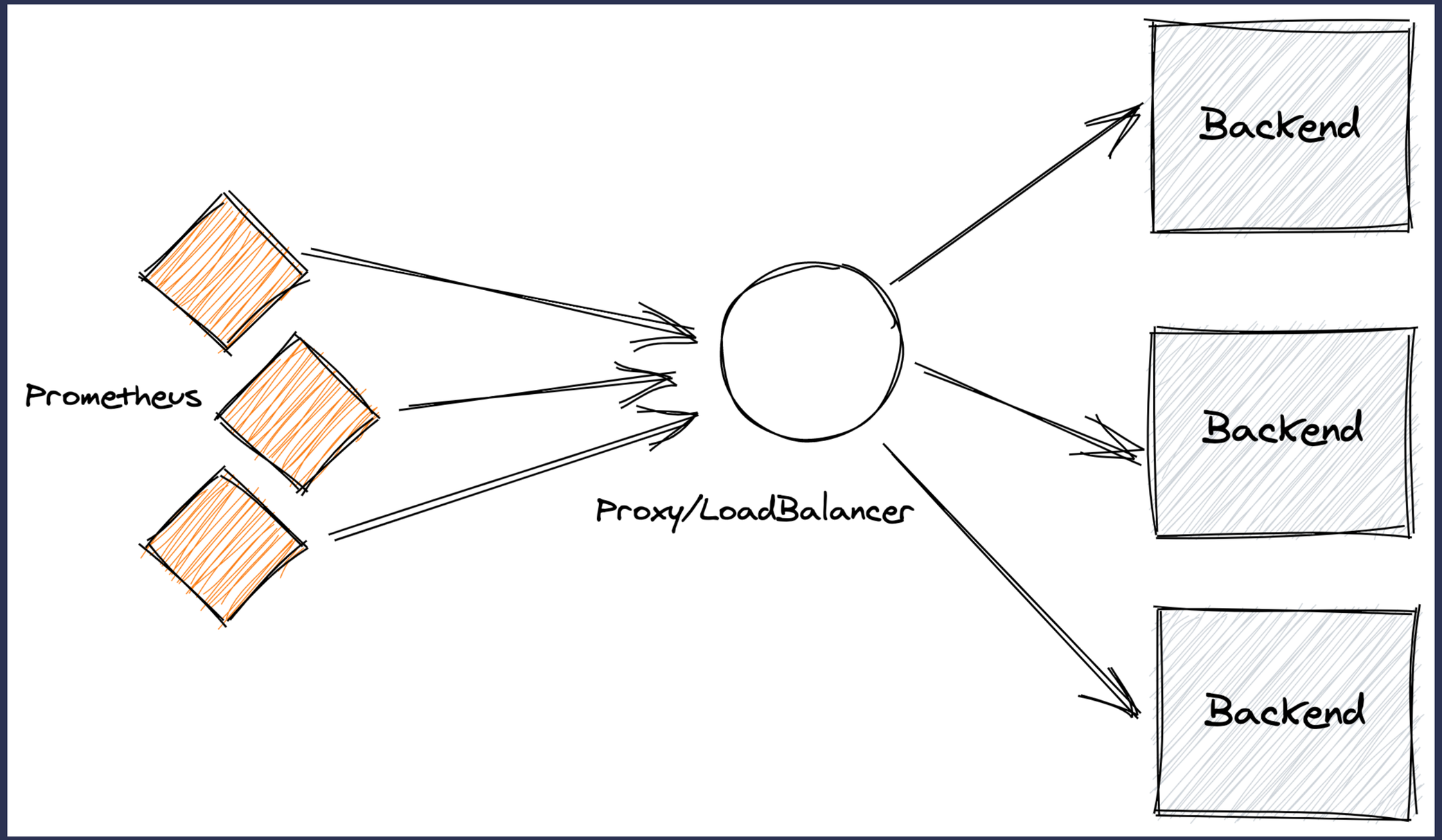
What does a properly observable system in Go look like?

The Pillars of Observability


Metrics
Tracing
Logging

Metrics, tracing, and logging

Everything starts with Instrumentation


Backend
Backend
Backend
Prometheus
Proxy/LoadBalancer

Health checks

```go
// NewServer creates a new internal server that exposes debug probes.
func NewServer(reg prometheus.Gatherer, listen, healthcheckURL string) *h
    // Internal server to expose introspection APIs.
    mux := http.NewServeMux()

    // Initialize health checks.
    healthchecks := healthcheck.NewHandler()

    // Register health check endpoints.
    mux.Handle("/-/healthy", http.HandlerFunc(healthchecks.LiveEndpoint))
    mux.Handle("/-/ready", http.HandlerFunc(healthchecks.ReadyEndpoint))

    return &http.Server{
        Addr:    listen,
        Handler: mux,
    }
}
```

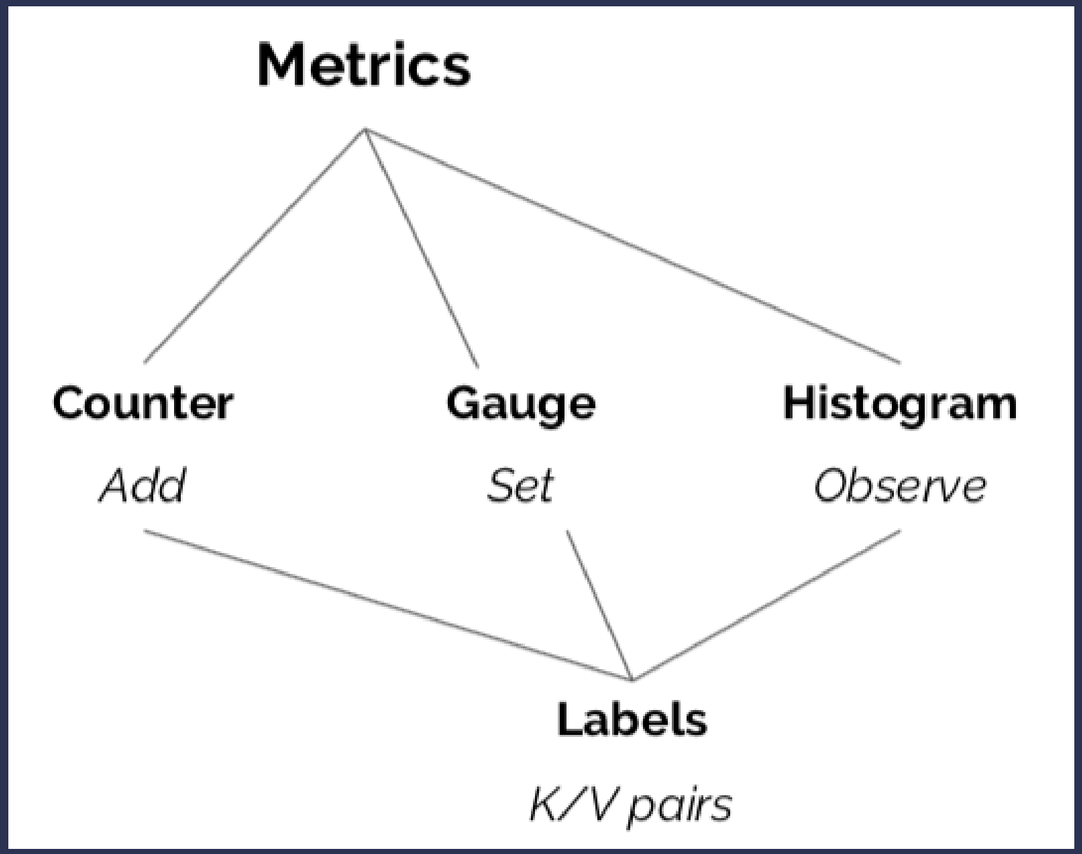
```
 curl localhost:8081/-/heathy
{
    "http": "OK"
}

 curl localhost:8081/-/ready
{
    "http": "OK"
}
```

metalmatze/signal

heptio/healthcheck

Metrics


Metrics
Counter
Add
Gauge
Set
Histogram
Observe
Labels
K/V pairs

Prometheus

Prometheus DOCS DOWNLOAD COMMUNITY SUPPORT & TRAINING BLOG Search

From metrics to insight

Power your metrics and alerting with a leading open-source monitoring solution.

GET STARTED DOWNLOAD

Dimensional data

Prometheus implements a highly dimensional data model. Time series are identified by a metric name and a set of key-value pairs.

Powerful queries

PromQL allows slicing and dicing of collected time series data in order to generate ad-hoc graphs, tables, and alerts.

Great visualization

Prometheus has multiple modes for visualizing data: a built-in expression browser, Grafana integration, and a console template language.

Efficient storage

Prometheus stores time series in memory and on local disk in an efficient custom format. Scaling is achieved by functional sharding and federation.

Simple operation

Each server is independent for reliability, relying only on local storage.

Precise alerting

Alerts are defined based on Prometheus's flexible PromQL and

Many client libraries

Client libraries allow easy instrumentation of services. Over ten

Many integrations

Existing exporters allow bridging of third-party data into Prometheus.

prometheus/client_golang

```go
package main

import "github.com/prometheus/client_golang/prometheus"

func main() {
    // Start metric registry.
    reg := prometheus.NewRegistry()

    // Register metrics server.
    http.Handle("/metrics", promhttp.HandlerFor(reg, promhttp.HandlerOpts
    http.ListenAndServe(":8080", nil)
}
```

```
 curl localhost:8081/metrics
# HELP go_gc_duration_seconds A summary of the pause duration of garbage
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
...
```

```go
import (
    "github.com/povilasv/prommod"
    "github.com/prometheus/common/version"
)

serviceName = "observable_remote_write_backend"

func main() {
    reg := prometheus.NewRegistry()

    // Register standard Go metric collectors, which are by default regis
    reg.MustRegister(
        prometheus.NewGoCollector(),
        version.NewCollector(serviceName),
        prommod.NewCollector(serviceName),
        prometheus.NewProcessCollector(prometheus.ProcessCollectorOpts{})
    )
```

Prometheus Alerts Graph Status Help

Enable query history

build_info

observable_remote_write_backend_**build_info**

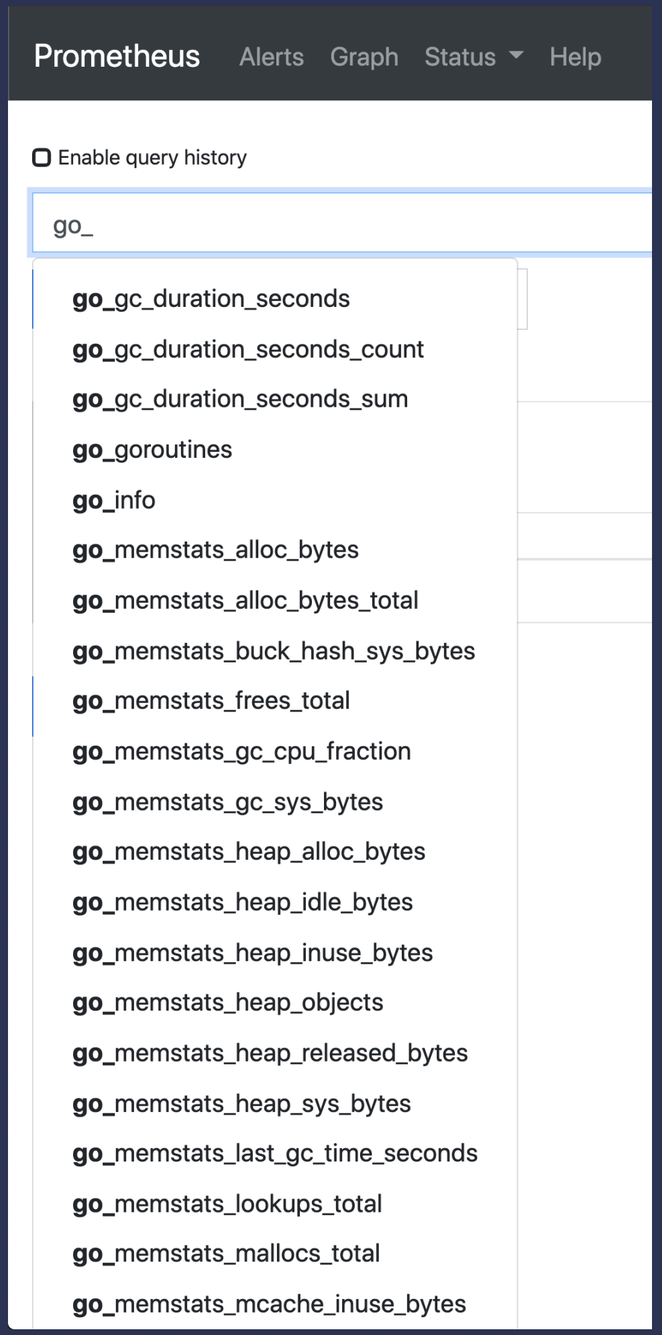
observable_remote_write_proxy_**build_info**

Moment

Element

no data

Add Graph


Prometheus
Alerts
Graph
Status
Help
Enable query history
go_
go_gc_duration_seconds
go_gc_duration_seconds_count
go_gc_duration_seconds_sum
go_goroutines
go_info
go_memstats_alloc_bytes
go_memstats_alloc_bytes_total
go_memstats_buck_hash_sys_bytes
go_memstats_frees_total
go_memstats_gc_cpu_fraction
go_memstats_gc_sys_bytes
go_memstats_heap_alloc_bytes
go_memstats_heap_idle_bytes
go_memstats_heap_inuse_bytes
go_memstats_heap_objects
go_memstats_heap_released_bytes
go_memstats_heap_sys_bytes
go_memstats_last_gc_time_seconds
go_memstats_lookups_total
go_memstats_mallocs_total
go_memstats_mcache_inuse_bytes

Prometheus Alerts Graph Status Help

Enable query history

Try experimental React UI

go_mod_info

Load time: 45ms
Resolution: 14s
Total time series: 93

Execute - insert metric at cursor -

Graph Console

Moment

Element	Value
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/apache/thrift",program="observable_remote_write_backend",version="v0.13.0"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/beorn7/perks",program="observable_remote_write_backend",version="v1.0.1"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/cespare/xxhash/v2",program="observable_remote_write_backend",version="v2.1.1"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/go-chi/chi",program="observable_remote_write_backend",version="v4.1.2+incompatible"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/go-kit/kit",program="observable_remote_write_backend",version="v0.10.0"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/go-logfmt/logfmt",program="observable_remote_write_backend",version="v0.5.0"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/gogo/protobuf",program="observable_remote_write_backend",version="v1.3.1"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/golang/protobuf",program="observable_remote_write_backend",version="v1.4.2"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/golang/snappy",program="observable_remote_write_backend",version="v0.0.1"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/grpc-ecosystem/grpc-gateway",program="observable_remote_write_backend",version="v1.14.6"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/matttproud/golang_protobuf_extensions",program="observable_remote_write_backend",version="v1.0.1"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/metalmatze/signal",program="observable_remote_write_backend",version="v0.0.0-20200616171423-be84551ba3ce"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/oklog/run",program="observable_remote_write_backend",version="v1.1.0"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/oklog/ulid",program="observable_remote_write_backend",version="v1.3.1"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/pkg/errors",program="observable_remote_write_backend",version="v0.9.1"}	1
go_mod_info{instance="127.0.0.1:8180",job="backend",name="github.com/povilasv/prommod",program="observable_remote_write_backend",version="v0.0.12"}	1

```go
package middleware

import (
    "github.com/prometheus/client_golang/prometheus"
)

type MetricsMiddleware struct {
    requestDuration *prometheus.HistogramVec
    requestSize     *prometheus.SummaryVec
    requestsTotal   *prometheus.CounterVec
    responseSize    *prometheus.SummaryVec
}
```

```
requestDuration: promauto.With(reg).NewHistogramVec(
    prometheus.HistogramOpts{
        Name:    "http_request_duration_seconds",
        Help:    "Tracks the latencies for HTTP requests.",
        Buckets: []float64{0.001, 0.01, 0.1, 0.3, 0.6, 1, 3, 6, 9, 20
    },
    []string{"code", "handler", "method"},
),

requestSize: promauto.With(reg).NewSummaryVec(
    prometheus.SummaryOpts{
        Name: "http_request_size_bytes",
        Help: "Tracks the size of HTTP requests.",
    },
    []string{"code", "handler", "method"},
),
```

```
import (
    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promauto"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

func (ins *MetricsMiddleware) NewHandler(handlerName string) func(next ht
    return func(handler http.Handler) http.Handler {
        return promhttp.InstrumentHandlerDuration(
            ins.requestDuration.MustCurryWith(prometheus.Labels{"handler"
            promhttp.InstrumentHandlerRequestSize(
                ins.requestSize.MustCurryWith(prometheus.Labels{"handler"
                    promhttp.InstrumentHandlerResponseSize(
                        ins.responseSize.MustCurryWith(prometheus.Labels{
                        handler,
                    ),
                ),
```

```
metrics := middleware.NewMetricsMiddleware(reg)

// Main server to listen for public APIs.
mux := http.NewServeMux()
mux.Handle("/receive",
    metrics.NewHandler("receive")(receiver.Receive),
)

srv := &http.Server{
    Addr:    cfg.server.listen,
    Handler: mux,
}

srv.ListenAndServe()
```

Prometheus Alerts Graph Status Help

Enable query history

Try experimental React UI

histogram_quantile(0.50, sum by (le) (rate(http_request_duration_seconds_bucket{job="observatorium-thanos-query",code!~"5..",handler="query_range"}[1d])))

Load time: 488ms
Resolution: 14s
Total time series: 1

Execute - insert metric at cursor ·

Graph Console

1h Until Res. (s) stacked

8.025
8.0249998
8.0249996
8.0249994

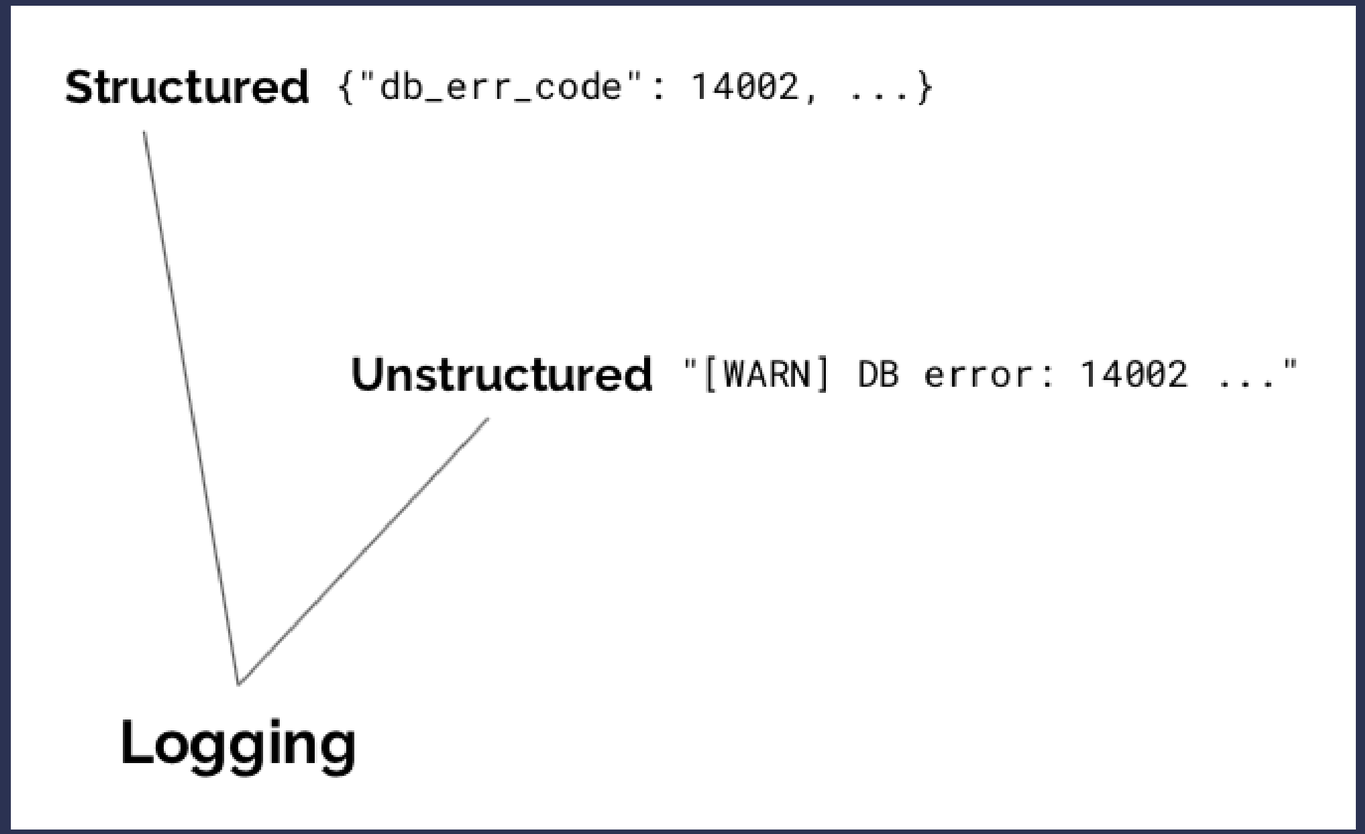
05:00 05:15 05:30 05:45

{}

Remove Graph

Add Graph

Logs


Structured {"db_err_code": 14002, ...}
Unstructured "[WARN] DB error: 14002 ..."
Logging

go-kit/logger

uber/zap

```go
func NewLogger(logLevel, logFormat, debugName string) log.Logger {
    var (
        logger log.Logger
        lvl    level.Option
    )

    switch logLevel {
    case "error":
        lvl = level.AllowError()
    case "warn":
        lvl = level.AllowWarn()
    case "info":
        lvl = level.AllowInfo()
    case "debug":
        lvl = level.AllowDebug()
    default:
        panic("unexpected log level")
```

```go
func main() {
    // Initialize structured logger.
    logger := internal.NewLogger(
        cfg.logLevel,
        cfg.logFormat,
        cfg.debug.name,
    )
    defer level.Info(logger).Log("msg", "exiting")
}
```

```
level.Warn(logger).Log("msg", "http read", "err", err)
level.Warn(logger).Log("msg", "snappy decode", "err", err)
level.Warn(logger).Log("msg", "proto unmarshalling", "err", err)
level.Info(logger).Log("msg", "remote write request received")
level.Debug(logger).Log("msg", m)
level.Debug(logger).Log("msg", fmt.Sprintf("  %f %d", s.Value, s.Timestam
```

```
level=info name=observable-remote-write-backend ts=2020-07-24T17:44:37Z
caller=main.go:127 msg="starting server"

level=info name=observable-remote-write-backend ts=2020-07-24T17:44:53Z
caller=receiver.go:64 msg="remote write request received"
```

```go
// Logger returns a middleware to log HTTP requests.
func Logger(logger log.Logger) func(next http.Handler) http.Handler {
    return func(next http.Handler) http.Handler {
        return http.HandlerFunc(func(w http.ResponseWriter, r *http.Reque
            start := time.Now()

            next.ServeHTTP(w, r)
            ...
            keyvals := []interface{}{
                "request", RequestIDFromContext(r.Context()),
                "proto", r.Proto,
                "method", r.Method,
                "path", r.URL.Path,
                "duration", time.Since(start),
            }

            level.Debug(logger).Log(keyvals...)
```

```
metrics := middleware.NewMetricsMiddleware(reg)

// Main server to listen for public APIs.
mux := http.NewServeMux()
mux.Handle("/receive",
    middleware.RequestID(
        middleware.Logger(logger)(receiver.Receive),
    ),
)

srv := &http.Server{
    Addr:    cfg.server.listen,
    Handler: mux,
}

srv.ListenAndServe()
```

```go
// RequestID returns a middleware that sets a unique request id for each
func RequestID(h http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request)
        reqID := r.Header.Get("X-Request-ID")
        if reqID == "" {
            entropy := ulid.Monotonic(rand.New(rand.NewSource(time.Now().
            reqID := ulid.MustNew(ulid.Timestamp(time.Now()), entropy)
            r.Header.Set("X-Request-ID", reqID.String())
        }
        ctx := newContextWithRequestID(r.Context(), reqID)
        h.ServeHTTP(w, r.WithContext(ctx))
    })
}
```

loki

Grafana Labs | Grafana | Products | Open Source | Learn | Downloads | Login | Contact us

Grafana | Cortex | Graphite | Loki | Metrictank | Prometheus | Tanka

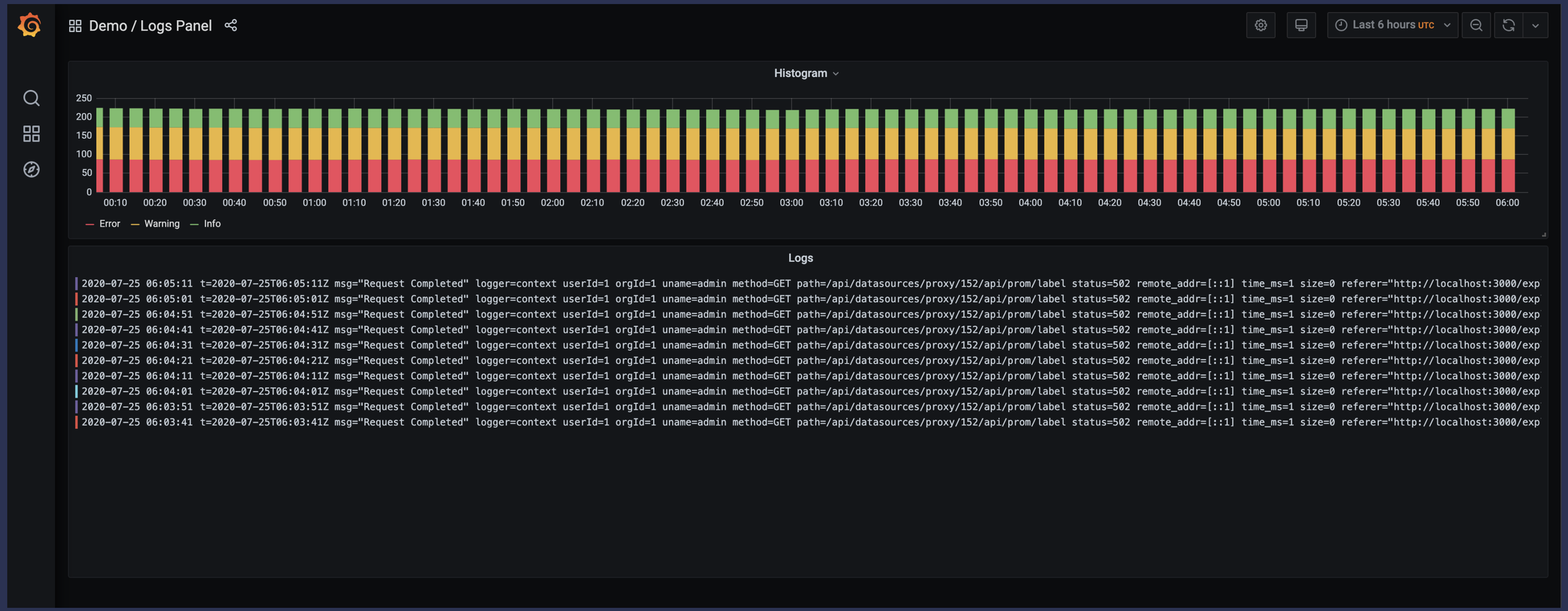
Loki

Loki is a horizontally-scalable, highly-available, multi-tenant log aggregation system inspired by Prometheus. It is designed to be very cost effective and easy to operate. It does not index the contents of the logs, but rather a set of labels for each log stream.

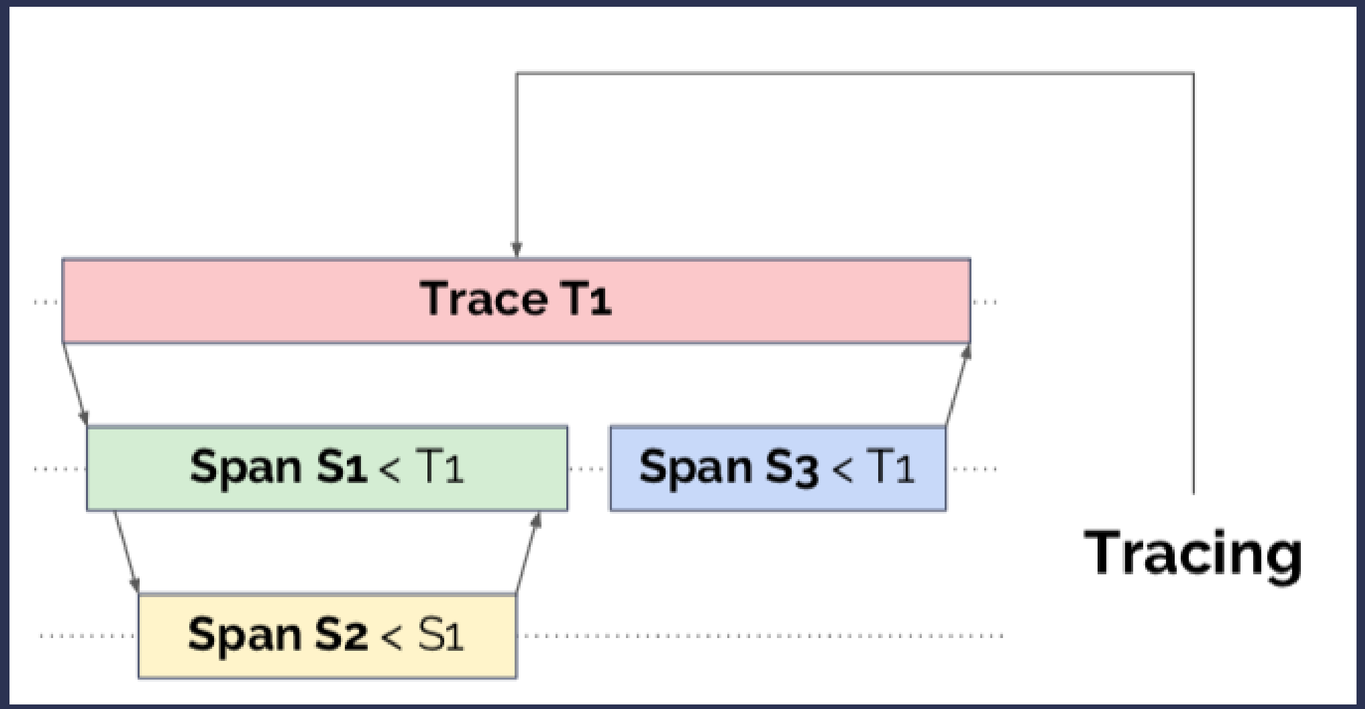
The Loki project was started at Grafana Labs in 2018, and announced at **KubeCon Seattle**. Loki is released under the Apache 2.0 License.

Grafana Labs is proud to lead the development of the Loki project, building first-class support for Loki into Grafana, and ensuring Grafana Labs customers receive Loki support and features they need.

- **Loki Documentation**
- **Loki Releases**
- **Loki Source**


Demo / Logs Panel
Last 6 hours UTC
Histogram
250
200
150
100
50
0
00:10 00:20 00:30 00:40 00:50 01:00 01:10 01:20 01:30 01:40 01:50 02:00 02:10 02:20 02:30 02:40 02:50 03:00 03:10 03:20 03:30 03:40 03:50 04:00 04:10 04:20 04:30 04:40 04:50 05:00 05:10 05:20 05:30 05:40 05:50 06:00
Error Warning Info
Logs
2020-07-25 06:05:11 t=2020-07-25T06:05:11Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp
2020-07-25 06:05:01 t=2020-07-25T06:05:01Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp
2020-07-25 06:04:51 t=2020-07-25T06:04:51Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp
2020-07-25 06:04:41 t=2020-07-25T06:04:41Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp
2020-07-25 06:04:31 t=2020-07-25T06:04:31Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp
2020-07-25 06:04:21 t=2020-07-25T06:04:21Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp
2020-07-25 06:04:11 t=2020-07-25T06:04:11Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp
2020-07-25 06:04:01 t=2020-07-25T06:04:01Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp
2020-07-25 06:03:51 t=2020-07-25T06:03:51Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp
2020-07-25 06:03:41 t=2020-07-25T06:03:41Z msg="Request Completed" logger=context userId=1 orgId=1 uname=admin method=GET path=/api/datasources/proxy/152/api/prom/label status=502 remote_addr=[::1] time_ms=1 size=0 referer="http://localhost:3000/exp

Tracing


Trace T1
Span S1 < T1
Span S3 < T1
Span S2 < S1
Tracing

jaeger

JAEGER

Download Docs Blog Project

Jaeger: open source, end-to-end distributed tracing

Monitor and troubleshoot transactions in complex distributed systems

GET STARTED DOWNLOAD

Why Jaeger?

As on-the-ground microservice practitioners are quickly realizing, the majority of operational problems that arise when moving to a distributed architecture are ultimately grounded in two areas: **networking** and **observability**. It is simply an orders of magnitude larger problem to network and debug a set of intertwined distributed services versus a single monolithic application.

Problems that Jaeger addresses

- distributed transaction monitoring
- performance and latency optimization
- root cause analysis
- service dependency analysis
- distributed context propagation

```
// Create and install Jaeger export pipeline.
traceProvider, closer, err := jaeger.NewExportPipeline(
    jaeger.WithCollectorEndpoint("http://127.0.0.1:14268/api/traces"),
    jaeger.WithProcess(jaeger.Process{
        ServiceName: serviceName,
        Tags: []kv.KeyValue{
            kv.String("exporter", "jaeger"),
        },
    }),
    jaeger.WithSDK(&sdktrace.Config{DefaultSampler: sdktrace.AlwaysSample
)
if err != nil {
    stdlog.Fatalf("failed to initialize tracer, err: %v", err)
}

defer closer()
```

```
func Receive(logger log.Logger, tracer trace.Tracer) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        ctx, span := tracer.Start(r.Context(), "receive")
        defer span.End()

        ...

        var reqBuf []byte

        if err := tracer.WithSpan(ctx, "decode", func(ctx context.Context
            var err error
            reqBuf, err = snappy.Decode(nil, compressed)
            return err
        }); err != nil {
            level.Warn(logger).Log("msg", "snappy decode", "err", err)
            http.Error(w, err.Error(), http.StatusBadRequest)
```

```
func Receive(logger log.Logger, tracer trace.Tracer) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        ctx, span := tracer.Start(r.Context(), "receive")
        defer span.End()

        ...

        var req prompb.WriteRequest

        if err := tracer.WithSpan(ctx, "unmarshal", func(ctx context.Cont
            return proto.Unmarshal(reqBuf, &req)
        }); err != nil {
            level.Warn(logger).Log("msg", "proto unmarshalling", "err", e
            http.Error(w, err.Error(), http.StatusBadRequest)

            return
        }
```

```
// Tracer returns an HTTP handler that injects the given tracer and start
// If any client span is fetched from the wire, we include that as our pa
func Tracer(logger log.Logger, tracer trace.Tracer, name string) func(nex
    operation := fmt.Sprintf("/%s HTTP[server]", name)
        ...
        return http.HandlerFunc(func(w http.ResponseWriter, r *http.Reque
            ctx := r.Context()
            attrs, entries, spanCtx := httptrace.Extract(ctx, r)
            r = r.WithContext(correlation.ContextWithMap(ctx, correlatior
                MultiKV: entries,
            }))

            ...
        })
    }
}
```

```
// Tracer returns an HTTP handler that injects the given tracer and start
// If any client span is fetched from the wire, we include that as our pa
func Tracer(logger log.Logger, tracer trace.Tracer, name string) func(nex
    operation := fmt.Sprintf("/%s HTTP[server]", name)
        ...
        return http.HandlerFunc(func(w http.ResponseWriter, r *http.Reque
            ...

            ctx, span := tracer.Start(
                trace.ContextWithRemoteSpanContext(ctx, spanCtx),
                name,
                trace.WithAttributes(attrs...),
            )
            defer span.End()

            span.AddEvent(ctx, operation)
```

Jaeger UI | Lookup by Trace ID... | Search | Compare | Dependencies | About Jaeger

Search | JSON File

Service (4)

thanos-query

Operation (25)

all

Tags

http.status_code=200 error=true

Lookback

Last Hour

Min Duration

e.g. 1.2s, 100ms, 500us

Max Duration

e.g. 1.2s, 100ms, 500us

Limit Results

20

Find Traces

Jaeger UI | Lookup by Trace ID... | Search | Compare | Dependencies | About Jaeger

Search | JSON File

Service (4)

thanos-query

Operation (25)

all

Tags

http.status_code=200 error=true

Lookback

Last Hour

Min Duration

e.g. 1.2s, 100ms, 500us

Max Duration

e.g. 1.2s, 100ms, 500us

Limit Results

20

Find Traces

20 Traces

Sort: Most Recent | Deep Dependency Graph

Compare traces by selecting result items

thanos-query: /query HTTP[server] 48de4eb — 2.11ms

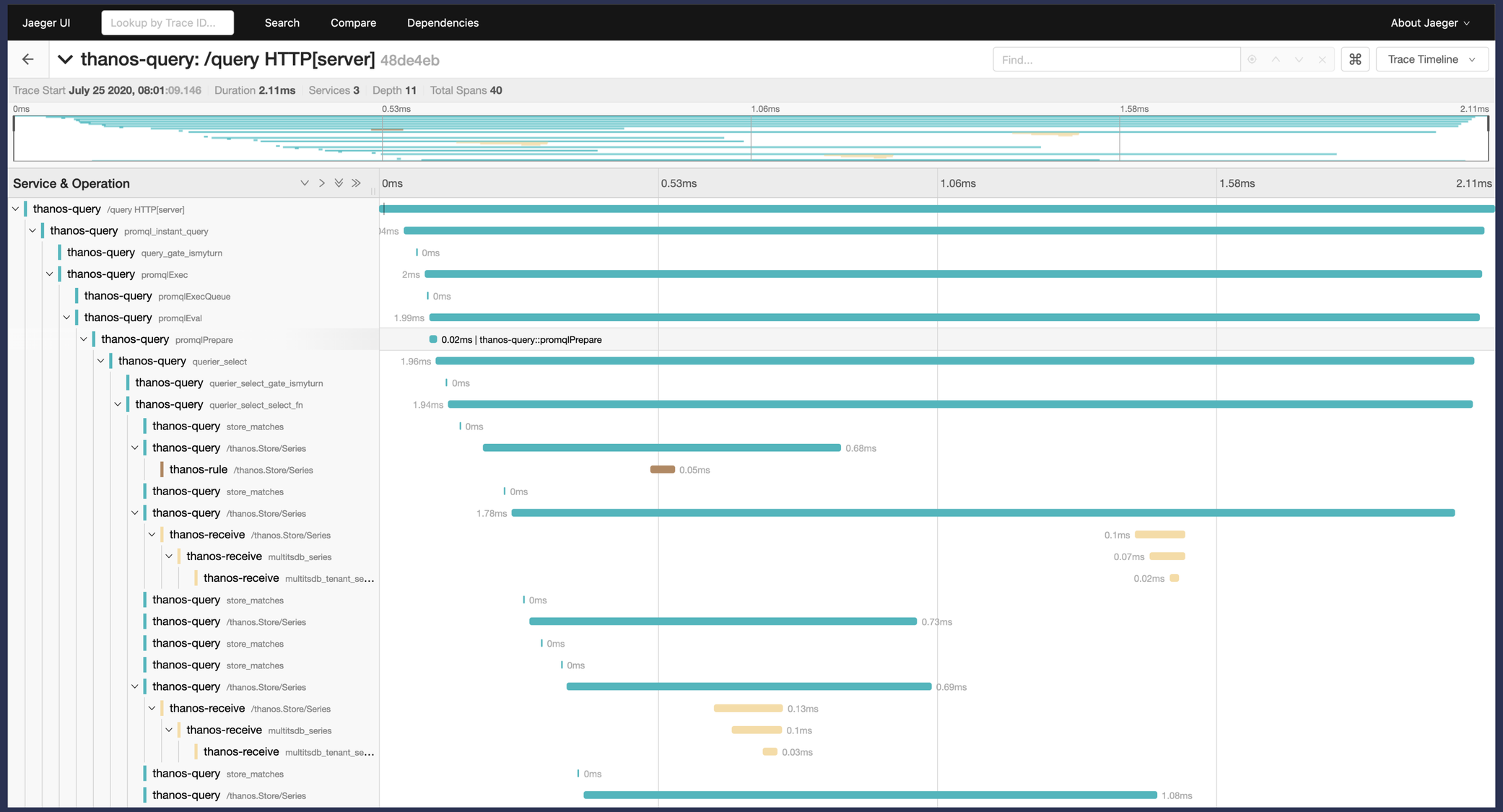
40 Spans | thanos-query (30) | thanos-receive (9) | thanos-rule (1) | Today | 8:01:09 am — a few seconds ago

thanos-query: /query HTTP[server] 7ad4f13 — 1.94ms

46 Spans | thanos-query (30) | thanos-receive (15) | thanos-rule (1) | Today | 8:01:08 am — a few seconds ago

thanos-query: /query HTTP[server] 3d465ed — 2.06ms

46 Spans | thanos-query (30) | thanos-receive (15) | thanos-rule (1) | Today | 8:01:08 am — a few seconds ago


Jaeger UI
Lookup by Trace ID...
Search
Compare
Dependencies
About Jaeger
thanos-query: /query HTTP[server] 48de4eb
Find...
Trace Timeline
Trace Start July 25 2020, 08:01:09.146 Duration 2.11ms Services 3 Depth 11 Total Spans 40
0ms
0.53ms
1.06ms
1.58ms
2.11ms
Service & Operation
thanos-query /query HTTP[server]
thanos-query promql_instant_query
thanos-query query_gate_ismyturn
thanos-query promqlExec
thanos-query promqlExecQueue
thanos-query promqlEval
thanos-query promqlPrepare
thanos-query querier_select
thanos-query querier_select_gate_ismyturn
thanos-query querier_select_select_fn
thanos-query store_matches
thanos-query /thanos.Store/Series
thanos-rule /thanos.Store/Series
thanos-query store_matches
thanos-query /thanos.Store/Series
thanos-receive /thanos.Store/Series
thanos-receive multitsdb_series
thanos-receive multitsdb_tenant_se...
thanos-query store_matches
thanos-query /thanos.Store/Series
thanos-query store_matches
thanos-query store_matches
thanos-query /thanos.Store/Series
thanos-receive /thanos.Store/Series
thanos-receive multitsdb_series
thanos-receive multitsdb_tenant_se...
thanos-query store_matches
thanos-query /thanos.Store/Series
4ms
0ms
2ms
0ms
1.99ms
0.02ms | thanos-query::promqlPrepare
1.96ms
0ms
1.94ms
0ms
0.68ms
0.05ms
0ms
1.78ms
0.1ms
0.07ms
0.02ms
0ms
0.73ms
0ms
0ms
0.69ms
0.13ms
0.1ms
0.03ms
0ms
1.08ms

14:00 - 15:00

Distributed Tracing on Go using OpenTelemetry

In the microservices era, monitoring applications become harder due to distributed and asynchronous data flow, therefore quickly identifying... Show More

Fahri Yardımcı

Profiling

```
func main() {
    debug := os.Getenv("DEBUG") != ""
    if debug {
        runtime.SetMutexProfileFraction(cfg.debug.mutexProfileFraction)
        runtime.SetBlockProfileRate(cfg.debug.blockProfileRate)
    }
}
```

```go
import "net/http/pprof"

// Internal server to expose introspection APIs.
mux := http.NewServeMux()

// Register pprof endpoints.
mux.HandleFunc("/debug/pprof/", pprof.Index)
mux.HandleFunc("/debug/pprof/cmdline", pprof.Cmdline)
mux.HandleFunc("/debug/pprof/profile", pprof.Profile)
mux.HandleFunc("/debug/pprof/symbol", pprof.Symbol)
mux.HandleFunc("/debug/pprof/trace", pprof.Trace)
```

localhost:8091/debug/pprof/

Red Hat | External | Ledger | Jira | Gitlab | Jenkins | Kibana | Incidents | Shiftzilla | Service Delivery /... | Observatorium | Docs

/debug/pprof/

Types of profiles available:

Count	Profile
30	allocs
0	block
0	cmdline
18	goroutine
30	heap
0	mutex
0	profile
18	threadcreate
0	trace

full goroutine stack dump

Profile Descriptions:

- allocs: A sampling of all past memory allocations
- block: Stack traces that led to blocking on synchronization primitives
- cmdline: The command line invocation of the current program
- goroutine: Stack traces of all current goroutines
- heap: A sampling of memory allocations of live objects. You can specify the gc GET parameter to run GC before taking the heap sample
- mutex: Stack traces of holders of contended mutexes
- profile: CPU profile. You can specify the duration in the seconds GET parameter. After you get the profile file, use the go tool pprof comm
- threadcreate: Stack traces that led to the creation of new OS threads
- trace: A trace of execution of the current program. You can specify the duration in the seconds GET parameter. After you get the trace fil

Profiling Go with pprof

Julia Evans

ABOUT
TALKS
PROJECTS
TWITTER
GITHUB

FAVORITES ★ ZINES ★ RSS

Profiling Go programs with pprof

Last week me and my cool coworker Josh were debugging some memory problems in a Go program using pprof.

There's a bunch of pprof documentation on the internet but I found a few things confusing so here are some notes so I can find them easily.

First – when working with pprof it's good to be running a recent version of Go! For example Go 1.8 adds mutex profiles so you can see mutex contention.

in this post I'll

- link to the useful pprof resource I found
- explain what a pprof profile is

Continues Profiling

Continues Profiling Go Programs

Continuous Profiling of Go programs

Jaana Dogan Following

Apr 26, 2018 · 4 min read

One of the most interesting parts of Google is our fleet-wide continuous profiling service. We can see who is accountable for CPU and memory usage, we can continuously monitor our production services for contention and blocking profiles, and we can generate analysis and reports and easily can tell what are some highly impactful optimization projects we can work on.

I briefly worked on Google Cloud Profiler, our new product that is filling the cloud-wide profiling gap for Cloud users. Note that you DON'T need to run your code on Google Cloud Platform in order to use it. Actually, I use it

Conprof

Why GitHub? Team Enterprise Explore Marketplace Pricing Search / Sign in Sign up

conprof / conprof

Watch 21 Star 280 Fork 20

Code Issues 9 Pull requests 3 Actions Security Insights

master 2 branches 0 tags Go to file Code

brancz Merge pull request #66 from yeya24/add-profile-download-api ... 3d56f13 11 days ago 125 commits

api	Add Profile Type	2 months ago
config	Add Profile Type	2 months ago
deployments	fixes after review	last month
examples	examples: Configure default config to scrape conprof itself	2 years ago
internal/trace	Add make sync	2 months ago
pkg/runutil	*: Fix versioning	12 months ago
pprofui	support downloading profiles	11 days ago
scrape	Add Profile Type	2 months ago
scripts	*: Fix versioning	12 months ago
vendor	*: Use go 1.13	9 months ago
web	Add Profile Type	2 months ago
.dockerignore	Add container make target	14 months ago
.drone.yml	Add arm build to latest	3 months ago
.gitattributes	Add generated git attribute	2 months ago
.gitignore	deployments: Rework to be more idiomatic jsonnet structure	9 months ago
.golangci.yml	*: Fix versioning	12 months ago
.promu.yml	*: Use go 1.13	9 months ago
Dockerfile	Add deployments/ dir containing kubernetes manifests	14 months ago
LICENSE	Initial commit	2 years ago

About

Continuous profiling in for pprof compatible profiles.

pprof profiling continuous-profiling profiles prometheus

Readme

Apache-2.0 License

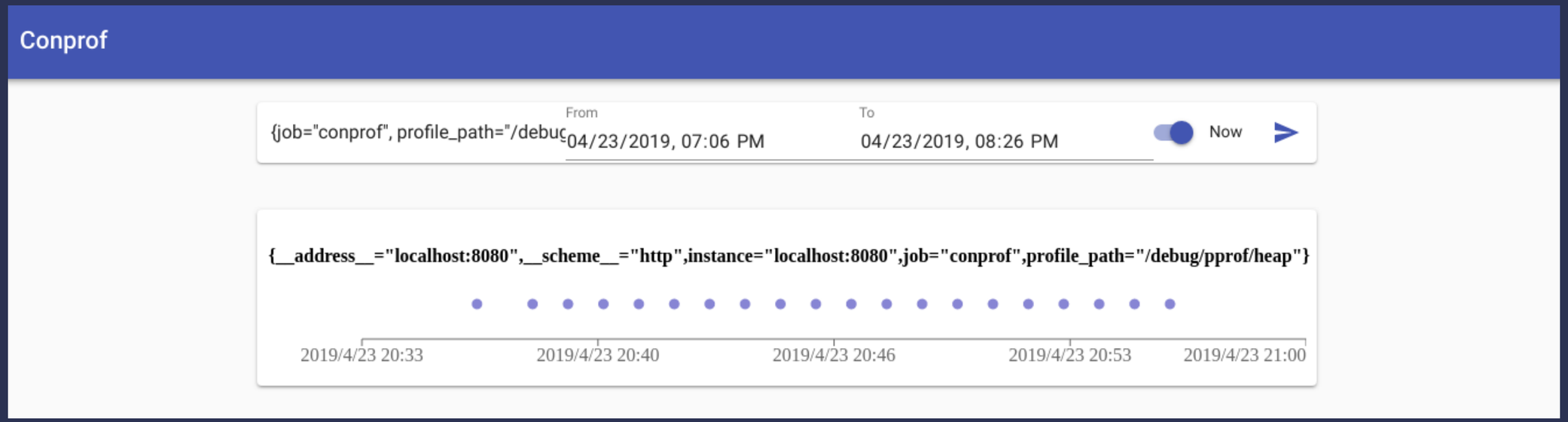
Releases

No releases published

Contributors 11

Languages

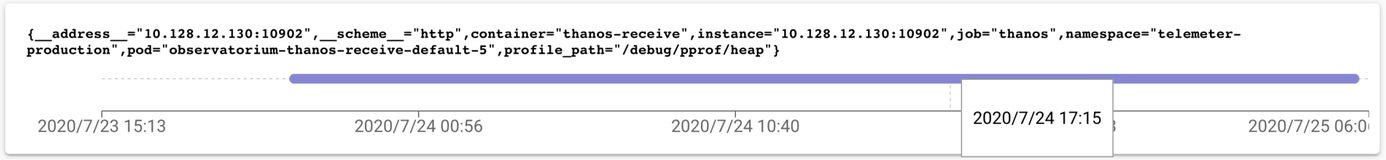
Go 69.6% TypeScript 15.6% Jsonnet 6.4% JavaScript 4.0% Makefile 2.2% HTML 1.8% Other 0.4%

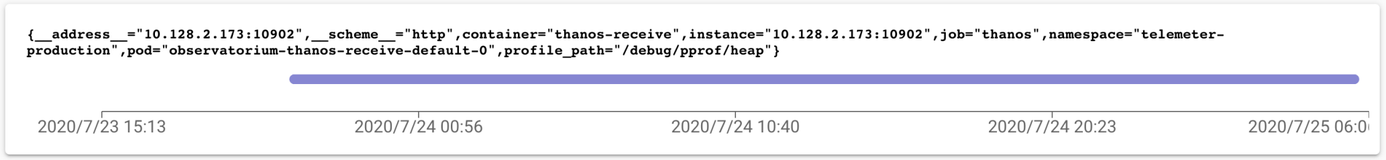

Conprof
{job="conprof", profile_path="/debug
From
04/23/2019, 07:06 PM
To
04/23/2019, 08:26 PM
Now
{__address__="localhost:8080",__scheme__="http",instance="localhost:8080",job="conprof",profile_path="/debug/pprof/heap"}
2019/4/23 20:33
2019/4/23 20:40
2019/4/23 20:46
2019/4/23 20:53
2019/4/23 21:00

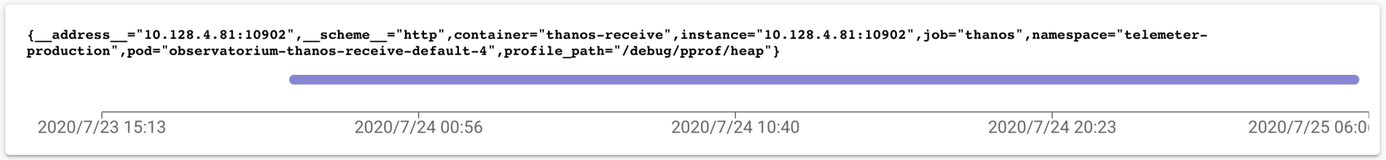
Conprof

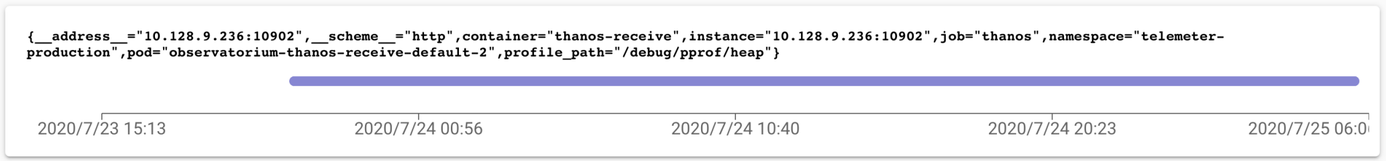
{container="thanos-receive",profile_path="/debug/pprof/heap"}

From 14.07.2020, 10:43 To 25.07.2020, 05:09 Now

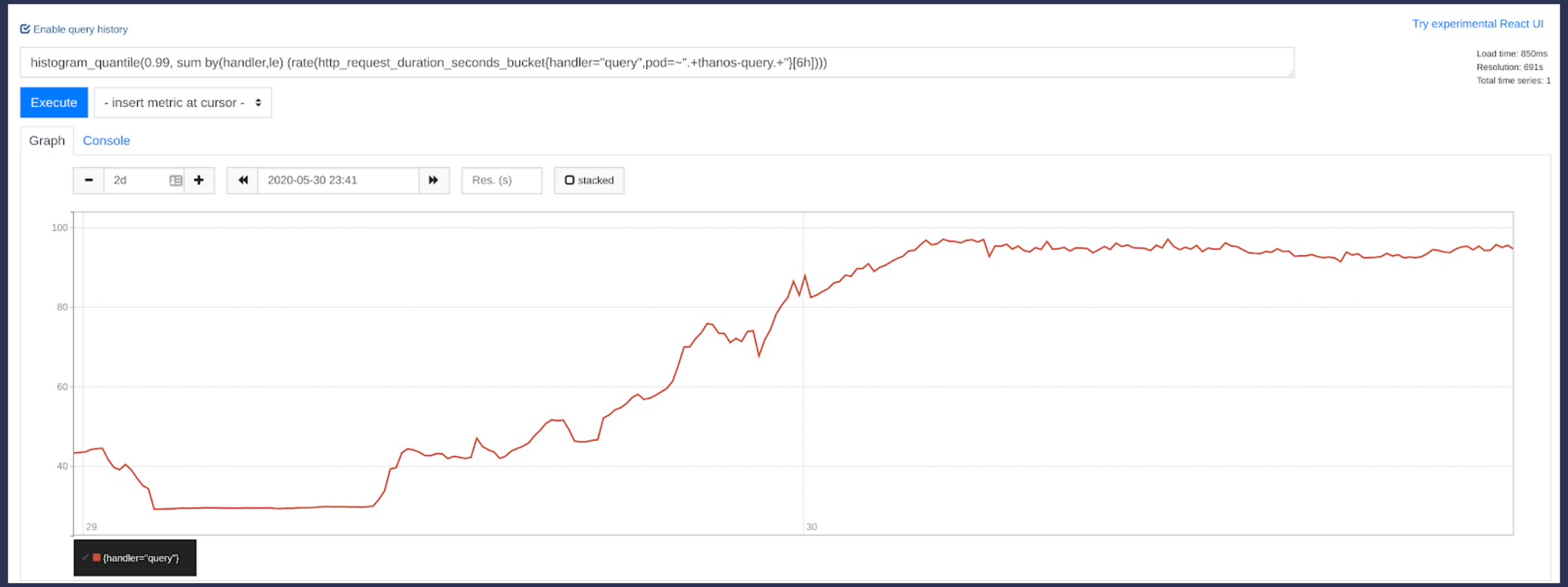






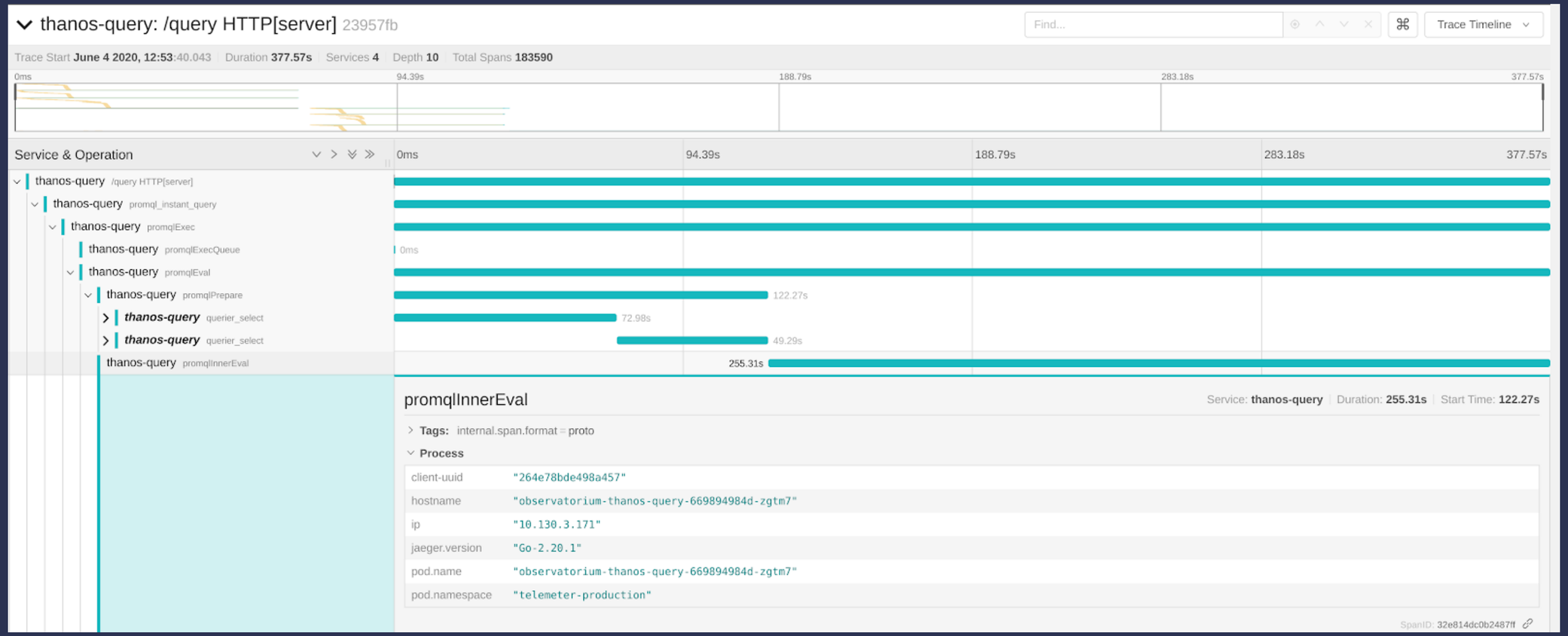


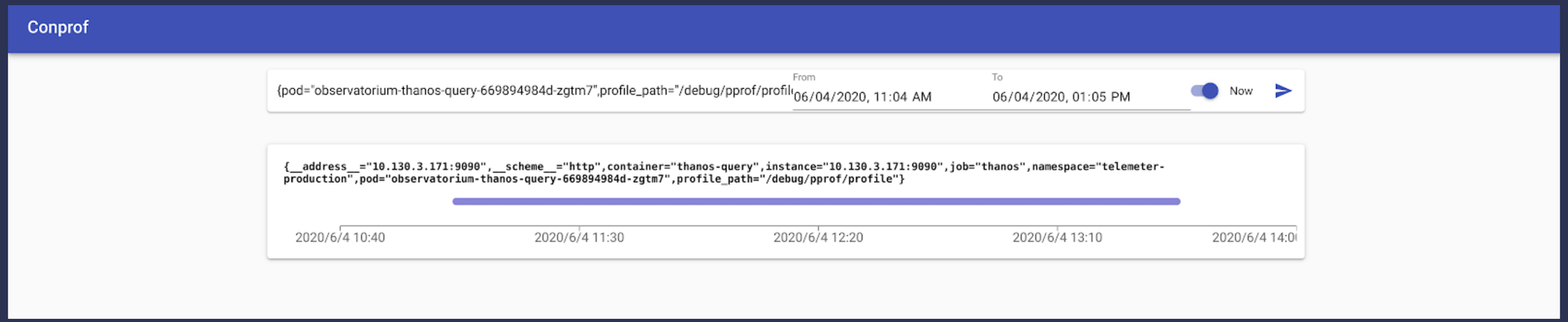
Correlation: Observability Superpower

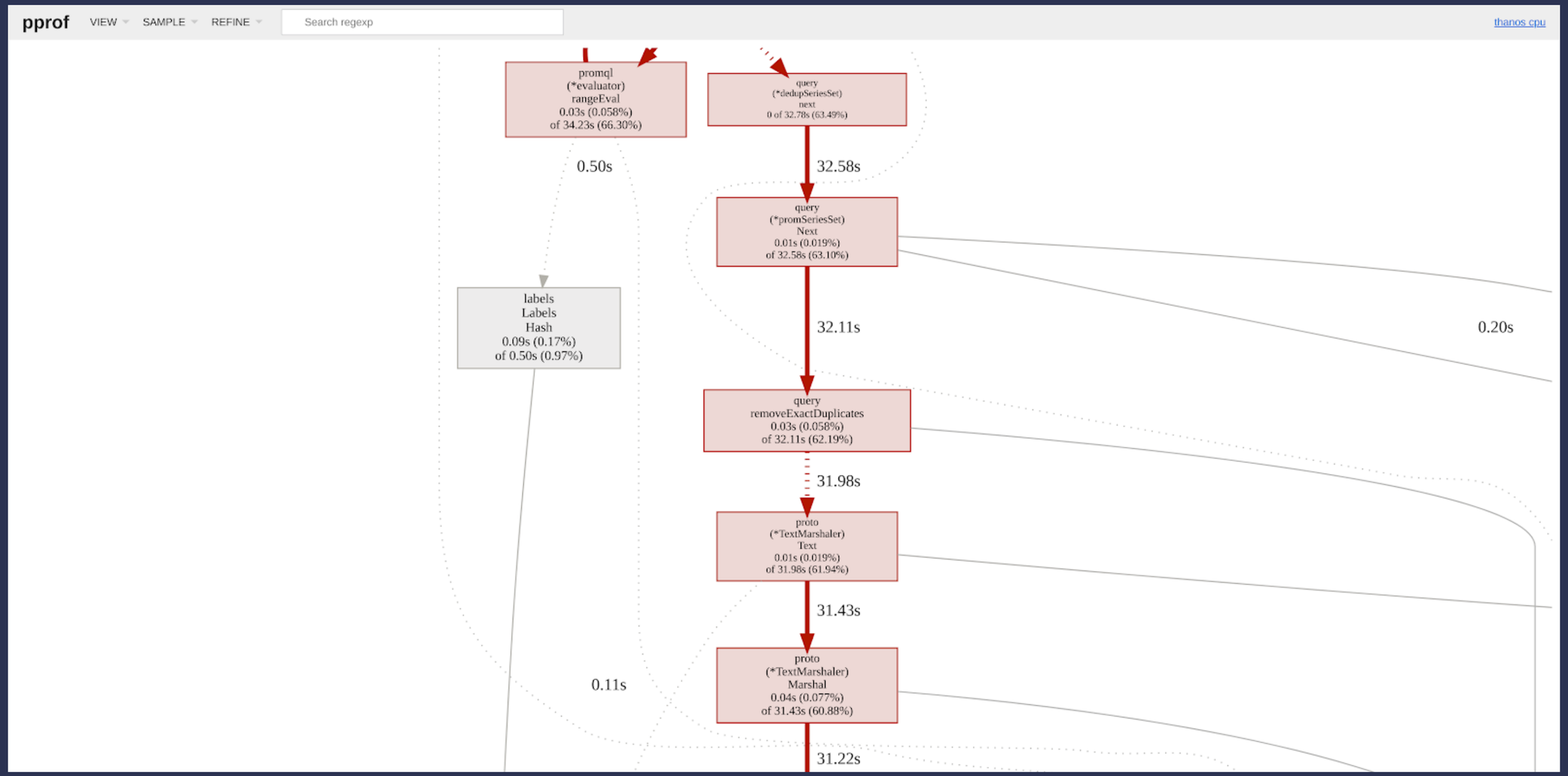

Enable query history
Try experimental React UI
histogram_quantile(0.99, sum by(handler,le) (rate(http_request_duration_seconds_bucket{handler="query",pod=~".+thanos-query.+"}[6h])))
Load time: 850ms
Resolution: 691s
Total time series: 1
Execute
- insert metric at cursor -
Graph
Console
2d
2020-05-30 23:41
Res. (s)
stacked
100
80
60
40
29
30
{handler="query"}

```
ts=2020-06-04T10:59:57.6170538Z component=query-reader msg="request finis
```

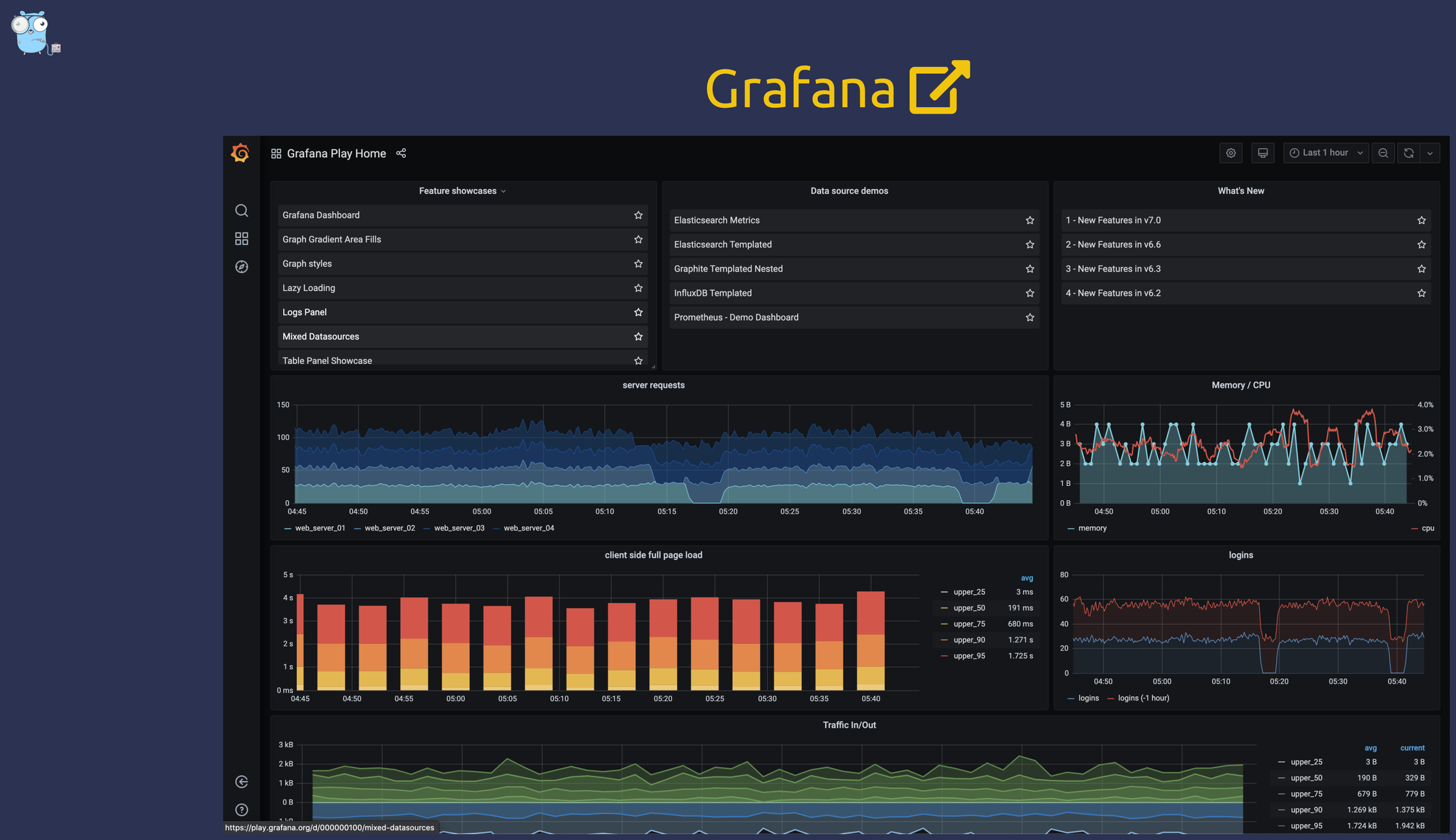
Log line from a slow query log, showing a trace ID from a slow query.


thanos-query: /query HTTP[server] 23957fb
Find...
Trace Timeline
Trace Start June 4 2020, 12:53:40.043 | Duration 377.57s | Services 4 | Depth 10 | Total Spans 183590
0ms
94.39s
188.79s
283.18s
377.57s
Service & Operation
thanos-query /query HTTP[server]
thanos-query promql_instant_query
thanos-query promqlExec
thanos-query promqlExecQueue
thanos-query promqlEval
thanos-query promqlPrepare
thanos-query querier_select
thanos-query querier_select
thanos-query promqlInnerEval
0ms
122.27s
72.98s
49.29s
255.31s
promqlInnerEval
Service: thanos-query | Duration: 255.31s | Start Time: 122.27s
Tags: internal.span.format = proto
Process
client-uuid "264e78bde498a457"
hostname "observatorium-thanos-query-669894984d-zgtm7"
ip "10.130.3.171"
jaeger.version "Go-2.20.1"
pod.name "observatorium-thanos-query-669894984d-zgtm7"
pod.namespace "telemeter-production"
SpanID: 32e814dc0b2487ff


Conprof
{pod="observatorium-thanos-query-669894984d-zgtm7",profile_path="/debug/pprof/profil
From
06/04/2020, 11:04 AM
To
06/04/2020, 01:05 PM
Now
{__address__="10.130.3.171:9090",__scheme__="http",container="thanos-query",instance="10.130.3.171:9090",job="thanos",namespace="telemeter-production",pod="observatorium-thanos-query-669894984d-zgtm7",profile_path="/debug/pprof/profile"}
2020/6/4 10:40
2020/6/4 11:30
2020/6/4 12:20
2020/6/4 13:10
2020/6/4 14:0


pprof
VIEW
SAMPLE
REFINE
Search regexp
thanos cpu
promql
(*evaluator)
rangeEval
0.03s (0.058%)
of 34.23s (66.30%)
query
(*dedupSeriesSet)
next
0 of 32.78s (63.49%)
0.50s
32.58s
query
(*promSeriesSet)
Next
0.01s (0.019%)
of 32.58s (63.10%)
labels
Labels
Hash
0.09s (0.17%)
of 0.50s (0.97%)
32.11s
0.20s
query
removeExactDuplicates
0.03s (0.058%)
of 32.11s (62.19%)
31.98s
proto
(*TextMarshaler)
Text
0.01s (0.019%)
of 31.98s (61.94%)
31.43s
0.11s
proto
(*TextMarshaler)
Marshal
0.04s (0.077%)
of 31.43s (60.88%)
31.22s

Grafana


Grafana
Grafana Play Home
Last 1 hour
Feature showcases
Grafana Dashboard
Graph Gradient Area Fills
Graph styles
Lazy Loading
Logs Panel
Mixed Datasources
Table Panel Showcase
Data source demos
Elasticsearch Metrics
Elasticsearch Templated
Graphite Templated Nested
InfluxDB Templated
Prometheus - Demo Dashboard
What's New
1 - New Features in v7.0
2 - New Features in v6.6
3 - New Features in v6.3
4 - New Features in v6.2
server requests
web_server_01
web_server_02
web_server_03
web_server_04
Memory / CPU
memory
cpu
client side full page load
avg
upper_25 3 ms
upper_50 191 ms
upper_75 680 ms
upper_90 1.271 s
upper_95 1.725 s
logins
logins
logins (-1 hour)
Traffic In/Out
avg
current
upper_25 3 B 3 B
upper_50 190 B 329 B
upper_75 679 B 779 B
upper_90 1.269 kB 1.375 kB
upper_95 1.724 kB 1.942 kB
https://play.grafana.org/d/000000100/mixed-datasources

prometheus-datasource

Grafana Labs | Grafana | Products | Open Source | Learn | Downloads | Login | Contact us

Blog | Case Studies | Community | Documentation | GrafanaCon 2020 | Tutorials

Prometheus data source

› Data sources › Prometheus

Grafana includes built-in support for Prometheus. This topic explains options, variables, querying, and other options specific to the Prometheus data source. Refer to Add a data source for instructions on how to add a data source to Grafana.

Prometheus settings

To access Prometheus settings, click the **Configuration** (gear) icon, then click **Data Sources**, and then click **Prometheus**.

Name	Description
Name	The data source name. This is how you refer to the data source in panels and queries.
Default	Default data source means that it will be pre-selected for new panels.
Url	The URL of your Prometheus server, e.g. `http://prometheus.example.org:9090` .
Access	Server (default) = URL needs to be accessible from the Grafana backend/server, Browser = URL needs to be accessible from the browser.
Basic Auth	Enable basic authentication to the Prometheus data source.
User	User name for basic authentication.
Password	Password for basic authentication.
Scrape interval	Set this to the typical scrape and evaluation interval configured in Prometheus. Defaults to 15s

loki-datasource

Grafana Labs | Grafana | Products | Open Source | Learn | Downloads | Login | Contact us

Blog | Case Studies | Community | Documentation | GrafanaCon 2020 | Tutorials

Using Loki in Grafana

› Data sources › Loki

> BETA: Querying Loki data requires Grafana's Explore section. Grafana v6.x comes with Explore enabled by default. In Grafana v5.3.x and v5.4.x. you need to enable Explore manually. Viewing Loki data in dashboard panels is supported in Grafana v6.4+.

Grafana ships with built-in support for Loki, Grafana's log aggregation system. Just add it as a data source and you are ready to query your log data in Explore.

Adding the data source

1. Open Grafana and make sure you are logged in.
2. In the side menu under the `Configuration` link you should find a link named `Data Sources`.
3. Click the `Add data source` button at the top.
4. Select `Loki` from the list of data sources.

> Note: If you're not seeing the `Data Sources` link in your side menu it means that your current user does not have the `Admin` role for the current organization.

Name	Description
Name	The data source name. This is how you refer to the data source in panels, queries, and Explore.
Default	Default data source means that it will be pre-selected for new panels.

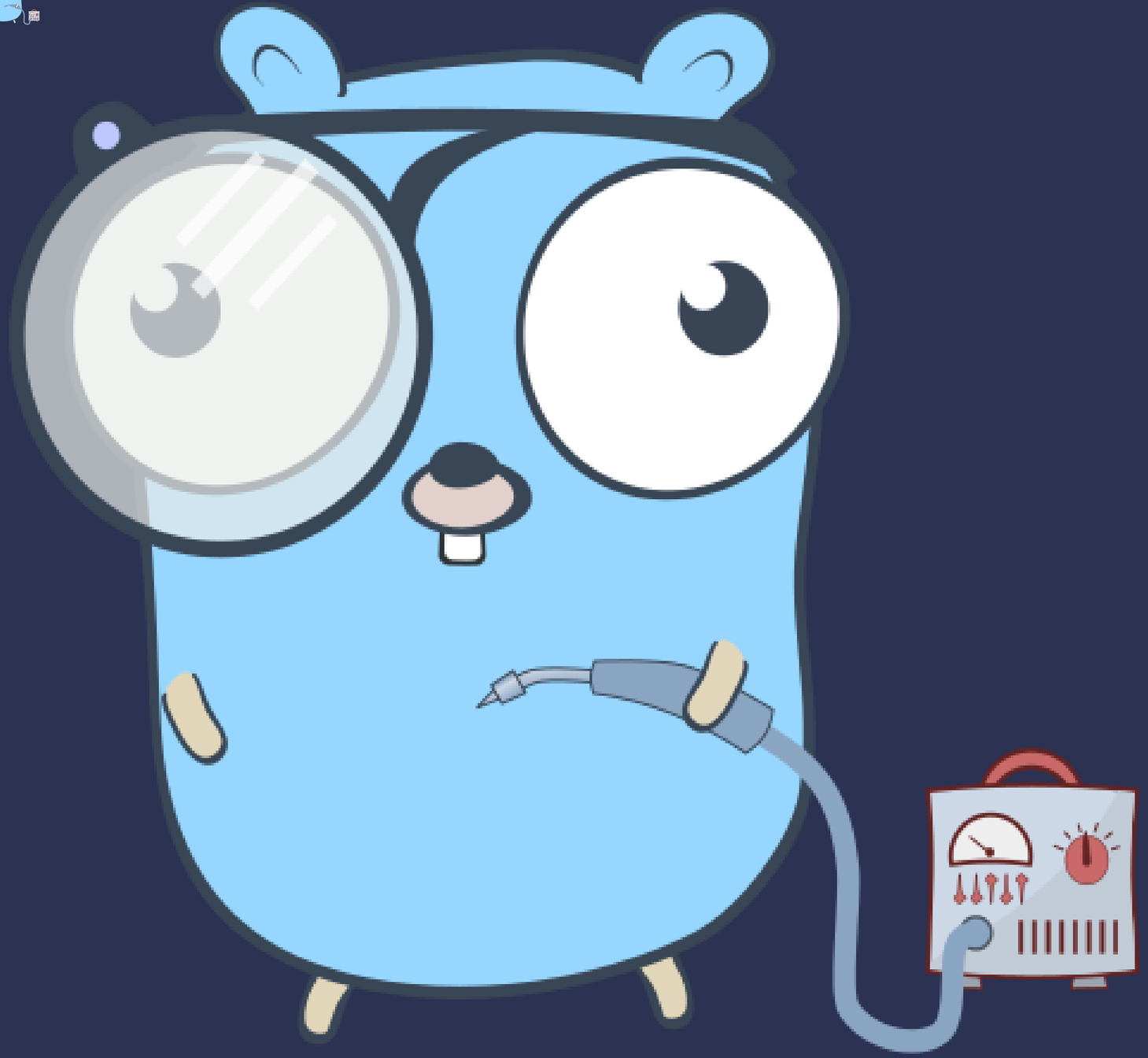
jaeger-datasource

Soon Conprof datasource

conprof-datasource

Check it yourself

observable-remote-write



Thanks a lot

Red Hat