

```
package main

import "fmt"

func main() {
    fmt.Println("Are you testing your Observability?")
    fmt.Println("    --- Metrics Edition ---    ")

    fmt.Println("    GoDays 22.01.2020, Berlin    ")
}
```



@bwplotka
@kakkoyun



Thanos

@bwplotka
@kakkoyun

HTTP
/lb

httputil.ReverseProxy.ServerHTTP(...)

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

1/3 HTTP
requests

1/3 HTTP
requests

1/3 HTTP
requests
@bwplotka
@kakkoyun

/lb

httputil.ReverseProxy.ServerHTTP(...)

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

$\frac{1}{3}$ HTTP
requests

$\frac{1}{3}$ HTTP
requests

$\frac{1}{3}$ HTTP
requests

@bwplotka
@kakkoyun

/lb

What's the % of requests finished with 502 HTTP code?

httputil.ReverseProxy.ServerHTTP(...)

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

1/3 HTTP requests

1/3 HTTP requests

1/3 HTTP requests

@bwplotka
@kakkoyun

/lb

httputil.ReverseProxv.ServerHTTP(...)

How many targets Discoverer actually returned at 5pm?

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

$\frac{1}{3}$ HTTP requests

$\frac{1}{3}$ HTTP requests

$\frac{1}{3}$ HTTP requests

@bwplotka
@kakkoyun

/lb

httputil.ReverseProxy.ServerHTTP(...)

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

How many time replica 2
was choosen?

$\frac{1}{3}$ HTTP
requests

Was it really $\frac{1}{3}$?

$\frac{1}{3}$ HTTP
requests

$\frac{1}{3}$ HTTP
requests

@bwplotka
@kakkoyun

/lb

What latency LB introduces
for most requests?

httputil.ReverseProxy.ServerHTTP(...)

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

Or maybe it's replica 1 that is slow?

1/3 HTTP
requests

1/3 HTTP
requests
@bwplotka
@kakkoyun

/lb

httputil.ReverseProxy.ServerHTTP(...)

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

How much memory LB actually used at 6pm?

1/3 HTTP
requests

1/3 HTTP
requests

1/3 HTTP
requests

@bwplotka
@kakkoyun

/lb

What version of LB was running on node X, two days ago?

httputil.ReverseProxy.ServerHTTP(...)

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

$\frac{1}{3}$ HTTP
requests

$\frac{1}{3}$ HTTP
requests

$\frac{1}{3}$ HTTP
requests

@bwplotka
@kakkoyun

HTTP

What version of LB was running on node X, two days ago?

What's the % of requests finished with 502 HTTP code?

httputil.ReverseProxy.ServerHTTP(...)

How many targets Discoverer actually returned at 5pm?

lbtransport.RoundTripper

Discoverer.Targets

How many time replica 2 was chosen?

RoundRobinPicker.Pick()

What latency LB introduces for most requests?

Or maybe it's target 3 that is slow?

http.RoundTripper

http.RoundTripper

http.RoundTripper

Was it really $\frac{1}{3}$?

How much memory LB actually used at 6pm?

HTTP
requests

$\frac{1}{3}$ HTTP
requests

Replica 1

Replica 2

Replica 3

@bwplotka
@kakkoyun

Let's instrument our LB with Prometheus metrics!

- Cheap
- Near Real Time
- Actionable (Alert-able)



<http://prometheus.io>

@bwplotka
@kakkoyun

/lb

What's the % of requests finished with 502 HTTP code?

httputil.ReverseProxy.ServerHTTP(...)

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

$\frac{1}{3}$ HTTP requests

$\frac{1}{3}$ HTTP requests

$\frac{1}{3}$ HTTP requests

@bwplotka
@kakkoyun

/lb

http_requests_total{code=...,method=...}

httputil.ReverseProxy.ServerHTTP(...)

lbtransport.RoundTripper

Discoverer.Targets()

RoundRobinPicker.Pick()

$\frac{1}{3}$ HTTP
requests

$\frac{1}{3}$ HTTP
requests

$\frac{1}{3}$ HTTP
requests

@bwplotka
@kakkoyun

Server HTTP request counter

```
// Increment our counter with written status code and request method.
serverRequestsTotal.WithLabelValues(statusRec.Status(), r.Method)).Add(1)
}

func init() {
    prometheus.MustRegister(serverRequestsTotal)
}

mux := http.NewServeMux()
mux.Handle("/metrics", promhttp.Handler())
mux.Handle("/lb", ...)

// Other handlers...

srv := &http.Server{Handler: mux}

// Run server...
```

Add /metrics handler.

@bwplotka
@kakkoyun

From code to graph

```
var (  
    serverRequestsTotal = prometheus.NewCounterVec(  
        prometheus.CounterOpts{  
            Name: "http_requests_total",  
            Help: "Tracks the number of HTTP requests.",  
        }, []string{"code", "method"},  
    )  
)  
  
// Part of response from /metrics HTTP endpoint.  
# HELP http_requests_total Tracks the number of HTTP requests.  
# TYPE http_requests_total counter  
http_requests_total{code="200", method="get"} 1089  
http_requests_total{code="500", method="get"} 46
```

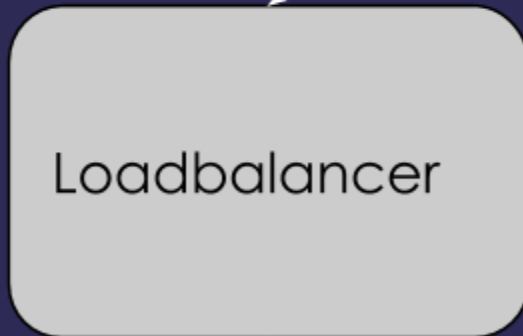
It is now exposed under /metrics.

@bwplotka
@kakkoyun



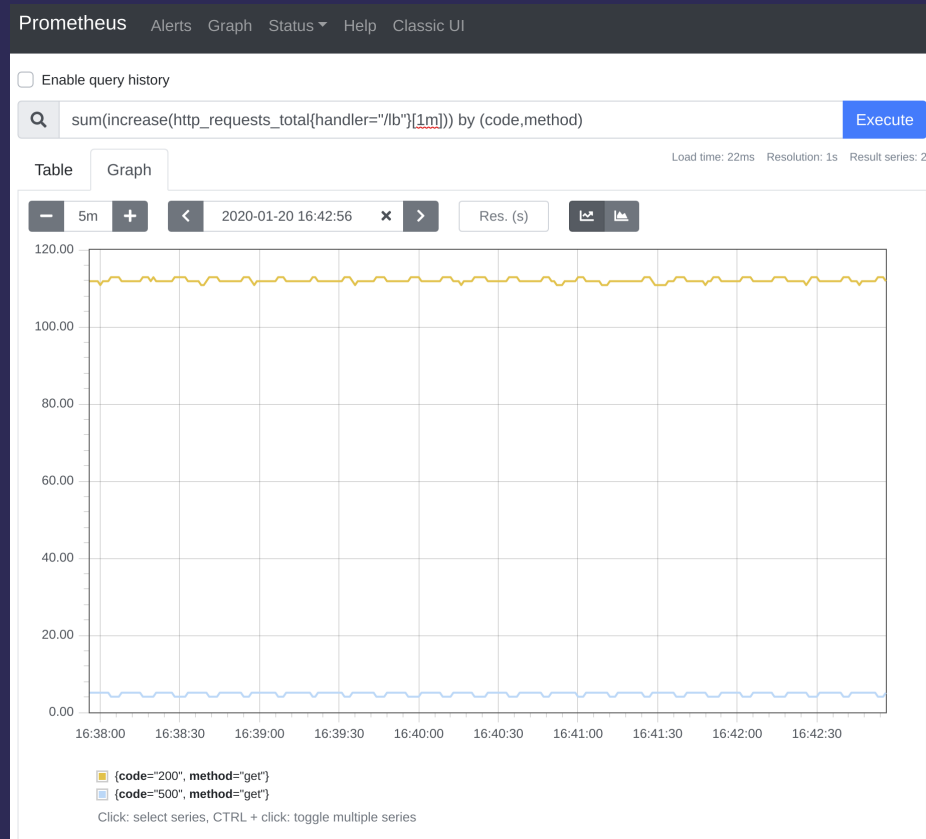
HTTP
/metrics

Periodic “scrape”



Loadbalancer

Graph: Prometheus UI





+



Pitfalls



@bwplotka
@kakkoyun

Pitfall #1: Global Registry

"magic is bad; global state is magic" by Peter Bourgon



@bwplotka
@kakkoyun

Pitfall #1: Getting rid of globals.

```
)
metrics3 := NewServerMetrics(
    prometheus.WrapWithLabels(prometheus.Labels{"handler":"/three"}, reg),
)

// For endpoint /one:
metrics1.requestsTotal.WithLabelValues(statusRec.Status(), r.Method).Add(1)

// For endpoint /two:
metrics2.requestsTotal.WithLabelValues(statusRec.Status(), r.Method).Add(1)

// For endpoint /three:
metrics3.requestsTotal.WithLabelValues(statusRec.Status(), r.Method).Add(1)

# HELP http_requests_total Tracks the number of HTTP requests.
# TYPE http_requests_total counter
http_requests_total{handler="/one", code="200", method="get"} 1445
http_requests_total{handler="/one", code="500", method="get"} 23
http_requests_total{handler="/two", code="200", method="get"} 445
http_requests_total{handler="/two", code="500", method="get"} 0
http_requests_total{handler="/three", code="200", method="get"} 645
http_requests_total{handler="/three", code="500", method="get"} 40
```

We can have request counter per handler (:

@bwplotka
@kakkoyun

Pitfall #2: No Tests For Metrics



Pitfall #2: No testing: Let's add tests!

```
// other cases...
} {
    if ok := t.Run("", func(t *testing.T) {
        // Prepare mocks...

        rec := httptest.NewRecorder()
        lb.ServeHTTP(
            rec,
            httptest.NewRequest("GET", "http://mocked", nil),
        )
        testutil.Equals(t, tcase.expectedCode, rec.Code)
        testutil.Equals(t, tcase.expectedHost, rec.Header().Get("X-lb-host"))

        // == RUN    TestLoadbalancer/#00
        //      --- FAIL: TestLoadbalancer/#00 (0.00s)
        //      transport_test.go:190:
        //
        //      exp: 1
        //
        //      got: 0
        //      FAIL
        //
        testutil.Equals(
            t,
            tcase.requestsCode502,
            promtestutil.ToFloat64(metrics.requestsTotal.WithLabelValues("502", "get")),
        )
        testutil.Equals(t, 1, promtestutil.CollectAndCount(lb.metrics.requestsTotal))

    }); !ok {
        return
    }
}
```

Now, test will catch our bug!

@bwplotka
@kakkoyun

Pitfall #3: Lack of Consistency

The Four Golden Signals, USE method, RED method etc...

- **R**: Requests per second (saturation).
- **E**: Errors per second.
- **D**: Duration (tail latency).

Pitfall #3: Consistency

```
type ServerMetrics struct {
    requestsTotal    *prometheus.CounterVec
    requestDuration  *prometheus.HistogramVec
}

// NewServerMetrics provides ServerMetrics.
func NewServerMetrics(reg prometheus.Registerer) *ServerMetrics {
    m := &ServerMetrics{
        requestsTotal: prometheus.NewCounterVec(
            prometheus.CounterOpts{
                Name: "http_requests_total",
                Help: "Tracks the number of HTTP requests.",
            }, []string{"code", "method"}),
        requestDuration: prometheus.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, 30, 60, 90, 120},
            }, []string{"code", "method"}),
    }
    reg.MustRegister(m.requestsTotal, m.requestDuration)
    return m
}

// Top level ServeHTTP handler.
func ServeHTTP(w http.ResponseWriter, r *http.Request) {
    start := time.Now()
    defer metrics.requestDuration.WithLabelValues(statusRec.Status(), r.Method).Observe(time.Since(start))

    statusRec := newStatusRecorder(w)
    next.ServeHTTP(statusRec, r)

    // Increment our counter with written status code and request method.
    metrics.requestsTotal.WithLabelValues(statusRec.Status(), r.Method).Inc()
}
```

Red method satisfied

@bwplotka
@kakkoyun

Pitfall #4: Naming: Not conforming naming convention

There is an official documentation on naming conventions

<https://prometheus.io/docs/practices/naming/#metric-and-label-naming>

@bwplotka
@kakkoyun

Pitfall #4: Naming: Not conforming naming convention

```
http_request_duration_seconds  
node_memory_usage_bytes
```

```
http_requests_total  
process_cpu_seconds_total
```

```
build_info
```

`_info` suffix for info metrics

@bwplotka
@kakkoyun

Pitfall #4: Naming: stability

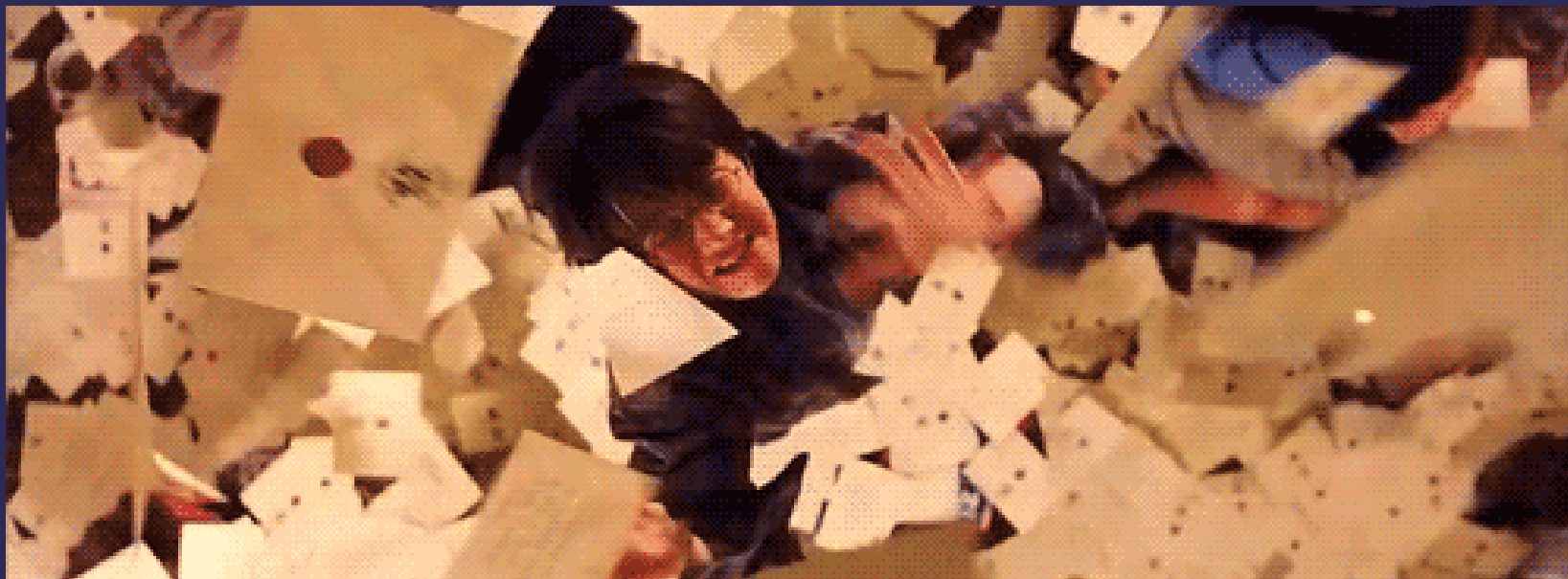
```
// NewServerMetrics provides ServerMetrics.
func NewServerMetrics(reg prometheus.Registerer) *ServerMetrics {
    m := &ServerMetrics{
        requestsTotal: prometheus.NewCounterVec(
            prometheus.CounterOpts{
                Name: "http_protocol_requests_total",
                Help: "Tracks the number of HTTP requests.",
            }, []string{"code", "method"},
        ),
    }
    reg.MustRegister(m.requestsTotal)
    return ins
}

## BOOM!💣 This alert will never fire but also will not fail!
## Rename can cause issues like this in Alerts, Recording rules, Dashboards and more..
alert: HttpToMany502Errors
  expr: |
    sum(rate(http_requests_total{status="502"}[1m])) /
    sum(rate(http_requests_total[1m])) * 100 > 5
  for: 5m
  labels:
    severity: error
  annotations:
    summary: "HTTP errors 502 (instance {{ $labels.instance }})"
    description: |
      "Too many HTTP requests with status 502 (> 5%)\n  VALUE = {{ $value }}\n
      LABELS: {{ $labels }}"
```

Ups...

@bwplotka
@kakkoyun

Pitfall #5: Cardinality: Unbounded metrics



```
# HELP conntrack_dialer_conn_failed_total Total number of connections failed to dial by the dialer.
# TYPE conntrack_dialer_conn_failed_total counter
conntrack_dialer_conn_failed_total{reason="<nil>"} 1
conntrack_dialer_conn_failed_total{reason="lookup example.com on localhost: err..."} 1
conntrack_dialer_conn_failed_total{reason="lookup thanos.io on 8.8.8.8: err..."} 1
conntrack_dialer_conn_failed_total{reason="lookup redhat.com on localhost: err..."} 1
conntrack_dialer_conn_failed_total{reason="syscall: unimplemented EpollWait"} 1
conntrack_dialer_conn_failed_total{reason="syscall.SIGINT: series of unfortunate things happened"} 1
conntrack_dialer_conn_failed_total{reason="unix: test returned fd in TestEpoll"} 1
conntrack_dialer_conn_failed_total{reason="Invalid value for argument: client: nil"} 1
```

Let's check out our metrics... 🕒

@bwplotka
@kakkoyun


```

        net.Conn: dialErrToReason(&failedConnRefused{err: err}).Msg()
        return conn, err
    }

    return &clientConnTracker{...}, nil
}

func dialErrToReason(err error) string {
    var e *net.OpError
    if errors.As(err, &e) {
        switch nestErr := e.Err.(type) {
        case *net.DNSError:
            return failedResolution
        case *os.SyscallError:
            if nestErr.Err == syscall.ECONNREFUSED {
                return failedConnRefused
            }

            return failedUnknown
        }

        if e.Timeout() {
            return failedTimeout
        }
    } else if err == context.Canceled || err == context.DeadlineExceeded {
        return failedTimeout
    }

    return failedUnknown
}

```

Pitfall #6: Cardinality: Histogram Explosion

```
# HELP http_request_duration_seconds Tracks the latencies for HTTP requests.
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.001"} 0
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.01"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.1"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.3"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.6"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="1"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="3"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="6"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="9"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="20"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="30"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="40"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="50"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="60"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="90"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="120"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="150"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="200"} 18
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="+Inf"} 18
http_request_duration_seconds_sum{code="200",handler="/metrics",method="get"} 0.0480237
http_request_duration_seconds_count{code="200",handler="/metrics",method="get"} 18
```



@bwplotka
@kakkoyun

[illegible]

@bwplotka
@kakkoyun

Pitfall #7: Poorly Chosen Histogram Buckets

```
# HELP http_request_duration_seconds Tracks the latencies for HTTP requests.
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.001"} 3
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.0012000000000000001"} 3
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.0014000000000000002"} 10
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.0016000000000000003"} 21
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.0018000000000000004"} 25
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="0.0020000000000000005"} 29
http_request_duration_seconds_bucket{code="200",handler="/metrics",method="get",le="+Inf"} 49
http_request_duration_seconds_sum{code="200",handler="/metrics",method="get"} 0.10939249999999999
http_request_duration_seconds_count{code="200",handler="/metrics",method="get"} 49
```

Pitfall #8: Not Initialized Metrics


```
m.connFailedTotal.WithLabelValues("unknown").Add(30)
```

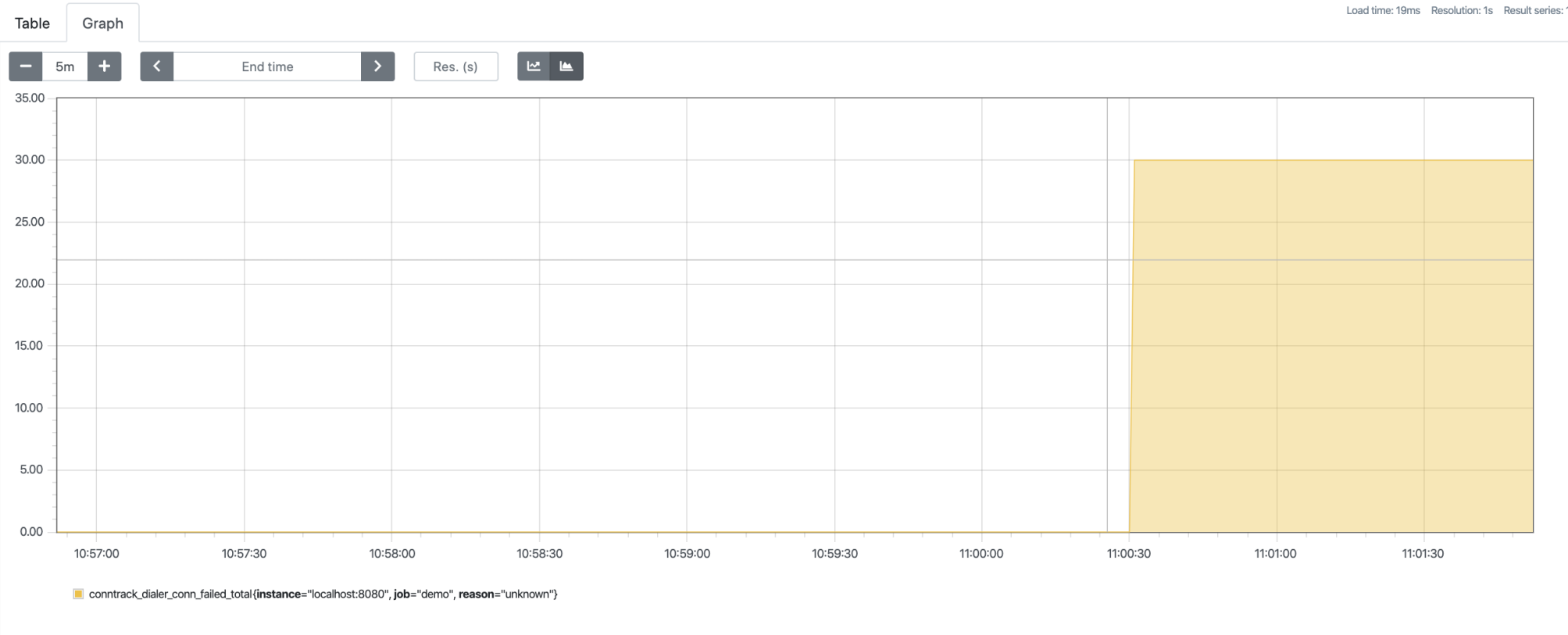
☒ Enable query history

Q

contrack_dialer_conn_failed_total

G

Execute

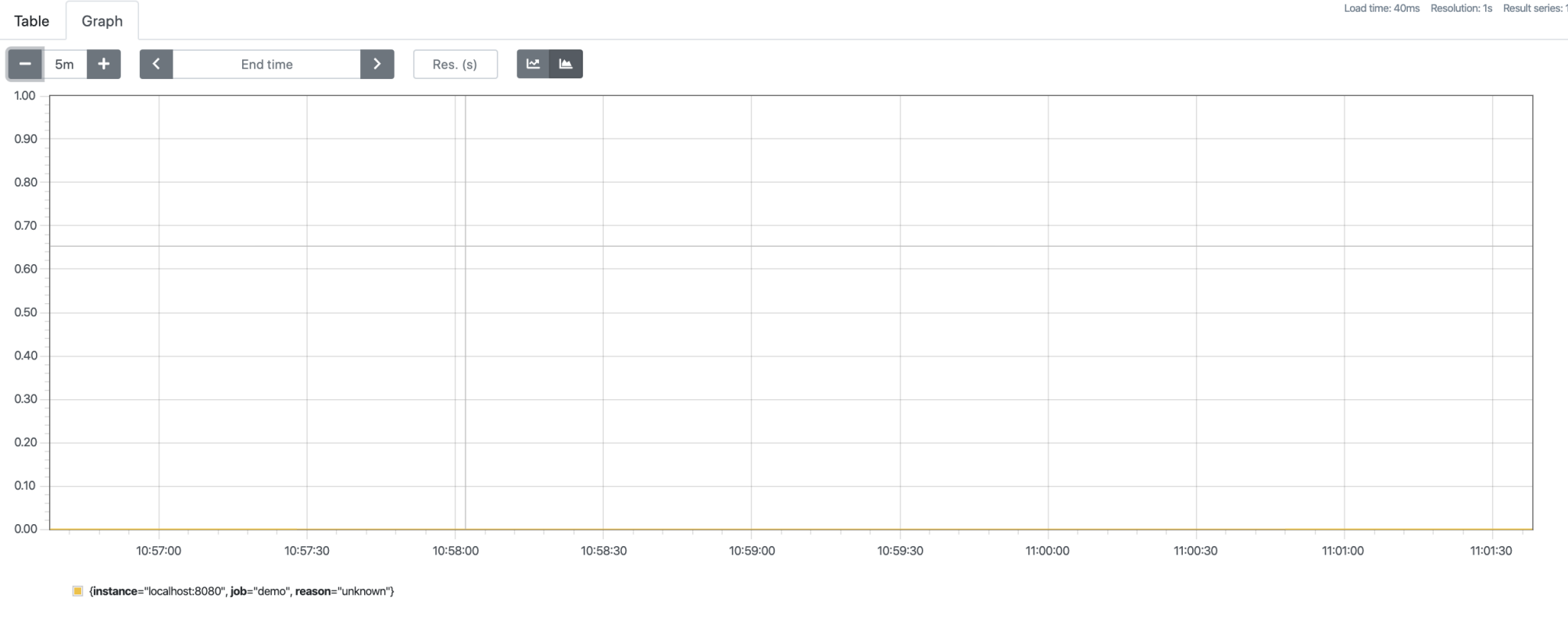


☒ Enable query history

Q

increase(contrack_dialer_conn_failed_total[1m])

Execute



[Remove Panel](#)

Not initializing Metrics

```
m.connFailedTotal.WithLabelValues("unknown").Add(30)
```

☒ Enable query history

Execute

Load time: 54ms Resolution: 3s Result series: 4

Table Graph

-

15m

+

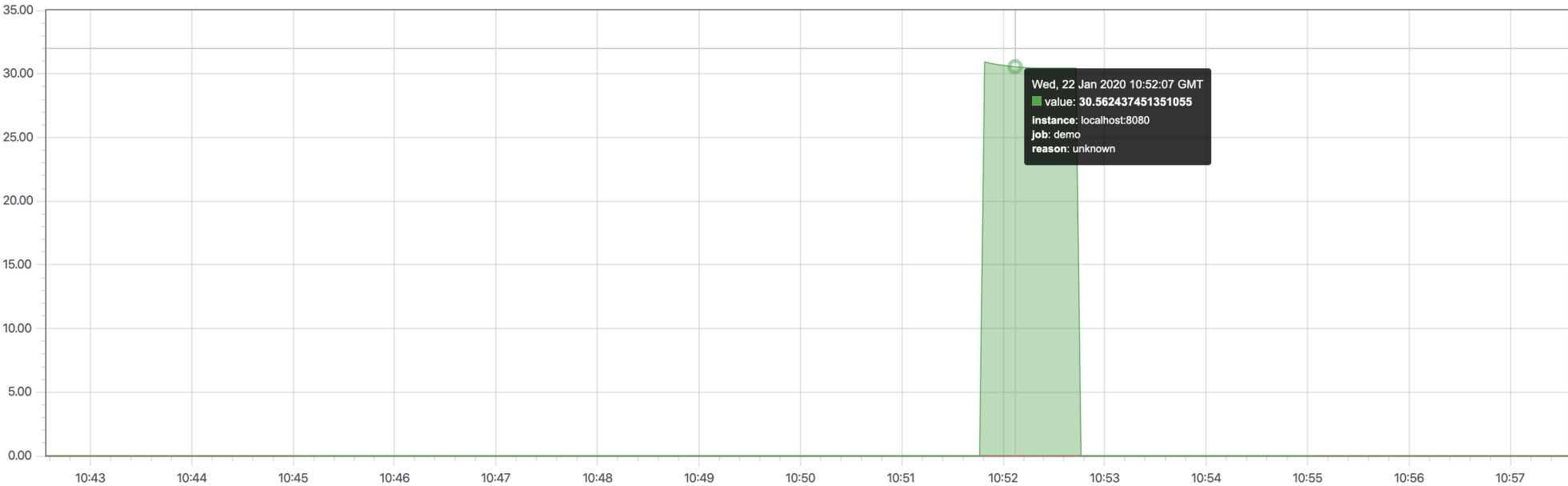
<

2020-01-22 10:57:34

×

>

Res. (s)



- {instance="localhost:8080", job="demo", reason="refused"}
- {instance="localhost:8080", job="demo", reason="resolution"}
- {instance="localhost:8080", job="demo", reason="timeout"}
- {instance="localhost:8080", job="demo", reason="unknown"}

Summary

Monitoring is essential.

Unit Test Your Instrumentation.

Avoid Global Registry.

<https://github.com/observatorium/observable-demo>

[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

observatorium / observable-demo

Unwatch

2

Unstar

4

Fork

1

[Code](#) [Issues 0](#) [Pull requests 0](#) [Actions](#) [Projects 0](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

Demo of the instrumented L7 HTTP loadbalancer in Go following best practices

[Manage topics](#)

12 commits

4 branches

0 packages

0 releases

2 contributors

Apache-2.0

Branch: master [New pull request](#) [Create new file](#) [Upload files](#) [Find file](#) [Clone or download](#)

kakkoyun and bwplotka Add diagram to README (#7) Latest commit fadac9a 40 minutes ago

cmd/loadbalancer	Added middlewares; fixes; added makefile demo command. (#3)	2 days ago
pkg	Add docs. (#5)	2 days ago
.drone.yml	Clean TODOs (#2)	3 days ago
.errcheck_excludes	Clean TODOs (#2)	3 days ago
.gitignore	Clean TODOs (#2)	3 days ago
.golangci.yml	Clean TODOs (#2)	3 days ago
Dockerfile	Add multistage container builds (#6)	23 hours ago
LICENSE	Initial commit	12 days ago
Makefile	Added middlewares; fixes; added makefile demo command. (#3)	2 days ago
README.md	Add diagram to README (#7)	40 minutes ago
demo-prometheus.yml	Added middlewares; fixes; added makefile demo command. (#3)	2 days ago
go.mod	Added test. (#4)	2 days ago
go.sum	Added middlewares; fixes; added makefile demo command. (#3)	2 days ago

README.md

observable-demo

semver-0.0.0 maintained yes build failure godoc reference go report A+

This repository includes simple L7 round robin loadbalancer implementation, instrumented with metrics.

WARNING

: This is not meant to be production ready L7 loadbalancer. It's missing things like proper retrying, DNS discovery, logging, tracing etc, that might be added in future (:

@bwplotka
@kakkoyun

Thank you!



Red Hat

<https://github.com/kakkoyun/are-you-testing-your-observability>

@bwplotka
@kakkoyun

Reference:

- [Thanos](#)
- [Prometheus](#)
- [Prometheus - client_go](#)
- [Prometheus - Histogram](#)
- [Prometheus Histograms - Past, Present, and Future](#)
- [Roboust Perception Blog](#)
- [Why_globals are magic](#)
- [Red Method](#)
- [Promcon 2019 - OpenMetrics](#)
- [Gopherize me](#)