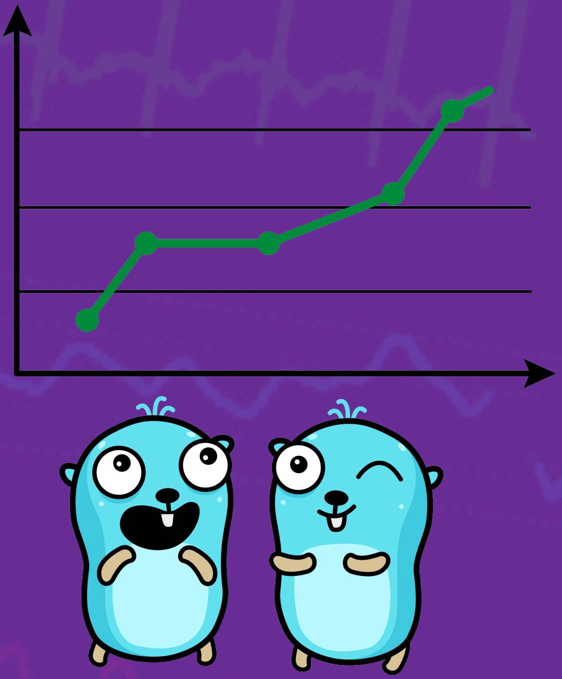


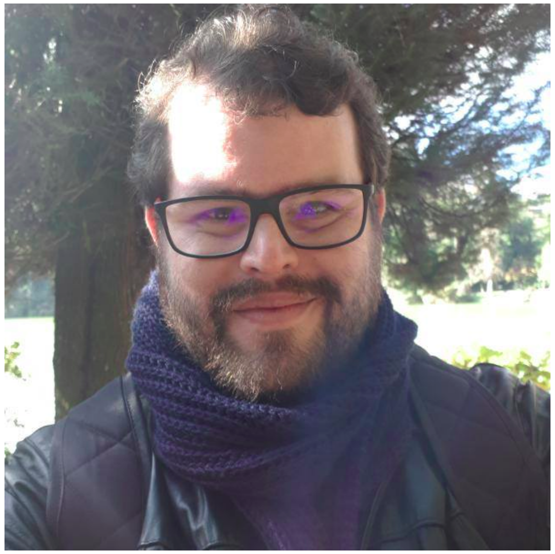
Monitore para Go e com Go

Monitore tudo com Go como meio ou como fim

Marco Paulo Ollivier

@marcopollivier

Marco Ollivier



Análise de Sistemas
@ Infnet

Software Engineer
@ OLX

Mentor de novos talentos
@ Codenation

Co-organizador
@ GopheRio

Palestrante

GopheRio

meetup.com/GopheRio

!!! CUIDADO !!!

Altíssimo nível de trocadilhos e analogias

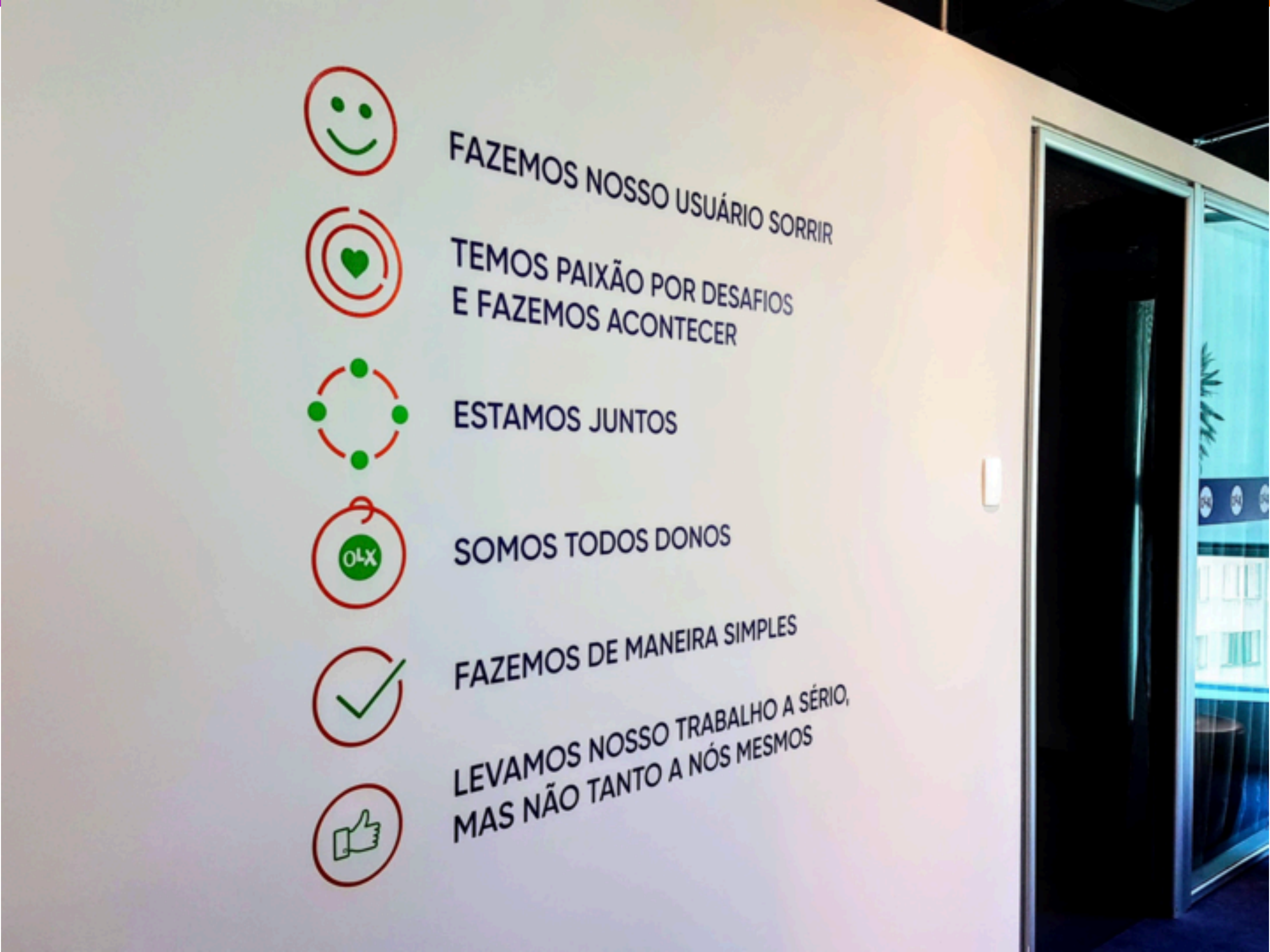
Agenda

- Porque monitorar é importante?
- Faça testes e exames preventivos
- Faça um eletrocardiograma
- Faça um checkup geral
- Diagnostique rápido, medique rápido
- Encerramento

Porque monitorar é importante?

Esteja sempre atendo, monitore e investigue quando algo parecer estranho

Usuários Felizes


FAZEMOS NOSSO USUÁRIO SORRIR
TEMOS PAIXÃO POR DESAFIOS E FAZEMOS ACONTECER
ESTAMOS JUNTOS
OLX
SOMOS TODOS DONOS
FAZEMOS DE MANEIRA SIMPLES
LEVAMOS NOSSO TRABALHO A SÉRIO, MAS NÃO TANTO A NÓS MESMOS

Clientes felizes são clientes pagantes

Muitas pessoas ainda compram a mentalidade "**se você construir, eles virão**”.

Se você colocar o **cliente em primeiro lugar**, ele continuará fiel ao seu aplicativo.

…uma das **piores coisas para o seu negócio é um site propenso a erros**.

Nada impulsionará os clientes se tiverem que esperar o site carregar.

Como 1seg custou US$1,6bi a Amazon

> *"Esse 1 segundo é essencial para fornecer uma grande experiência ao usuário" – Jeff Bezos, CEO da Amazon*

danielscott.com.br/como-1-segundo-custava-16-bilhao-em-vendas-a-amazon/

Como 1seg custou US$1,6bi a Amazon

Uma análise mais aprofundada mostra que a cada 100 milissegundos de demora, as vendas caem em 1%

danielscott.com.br/como-1-segundo-custava-16-bilhao-em-vendas-a-amazon/

Quanto vale 1/4 de seg para o Google?

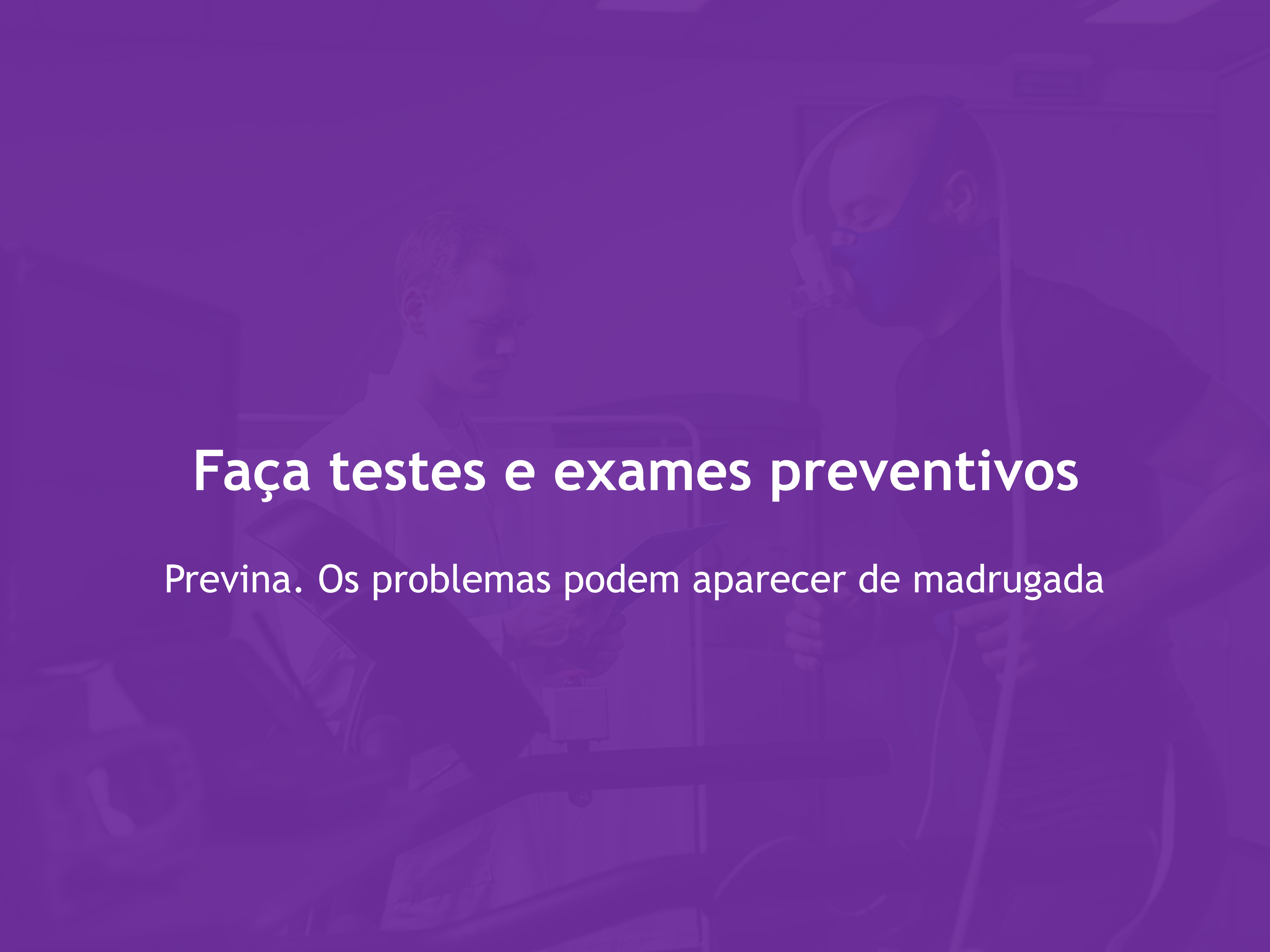
O Google estima que **1/4 de segundo** a mais para carregar uma busca resultaria numa perda de **8 milhões de buscas** por dia

danielscott.com.br/como-1-segundo-custava-16-bilhao-em-vendas-a-amazon/

4:00

Regra de ouro #1

Ache o problema antes do usuário

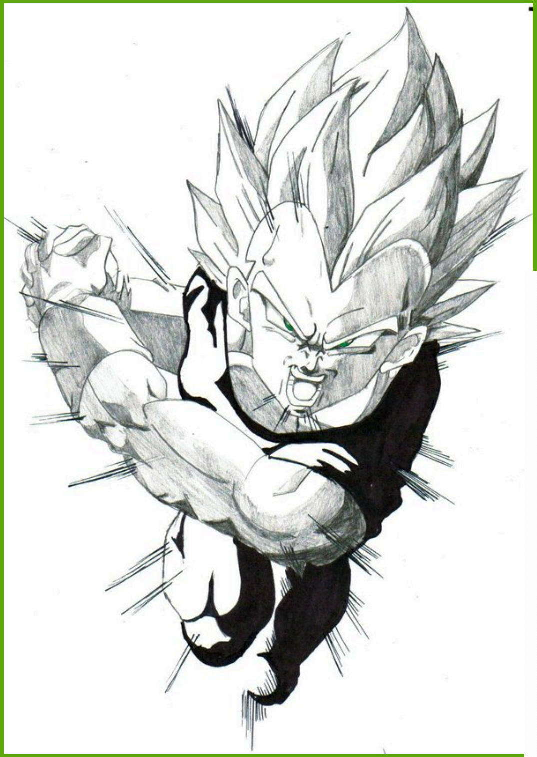


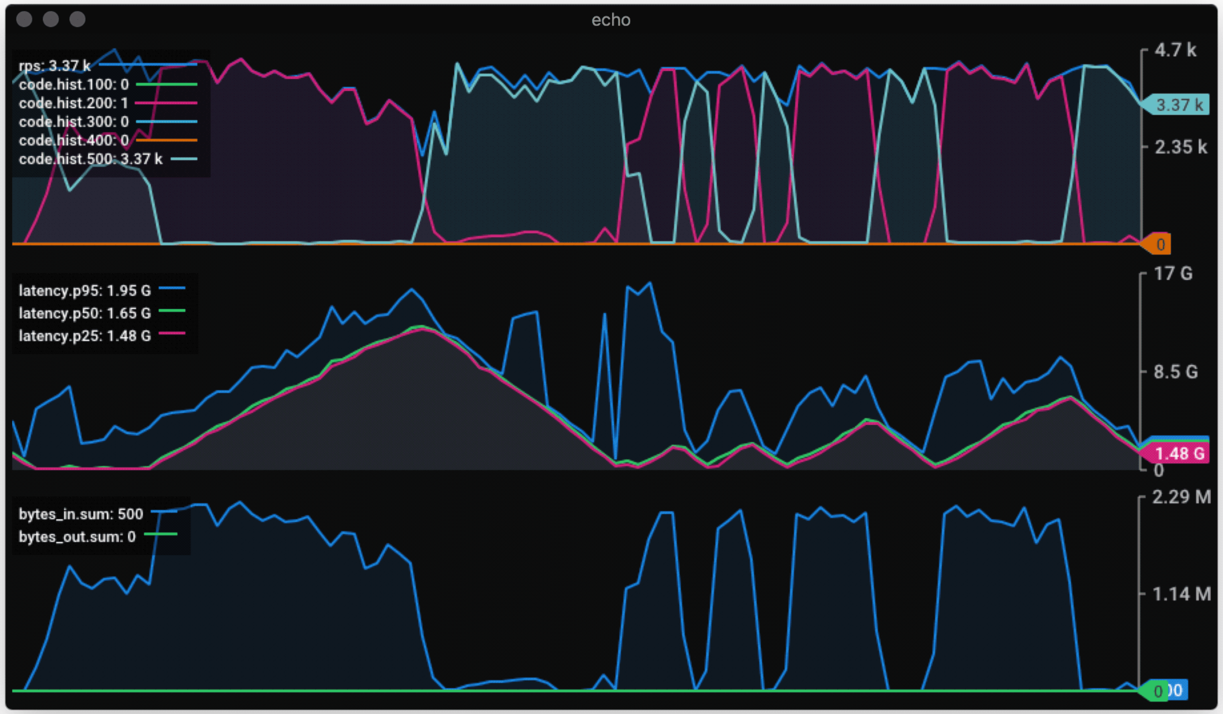
Faça testes e exames preventivos

Previna. Os problemas podem aparecer de madrugada

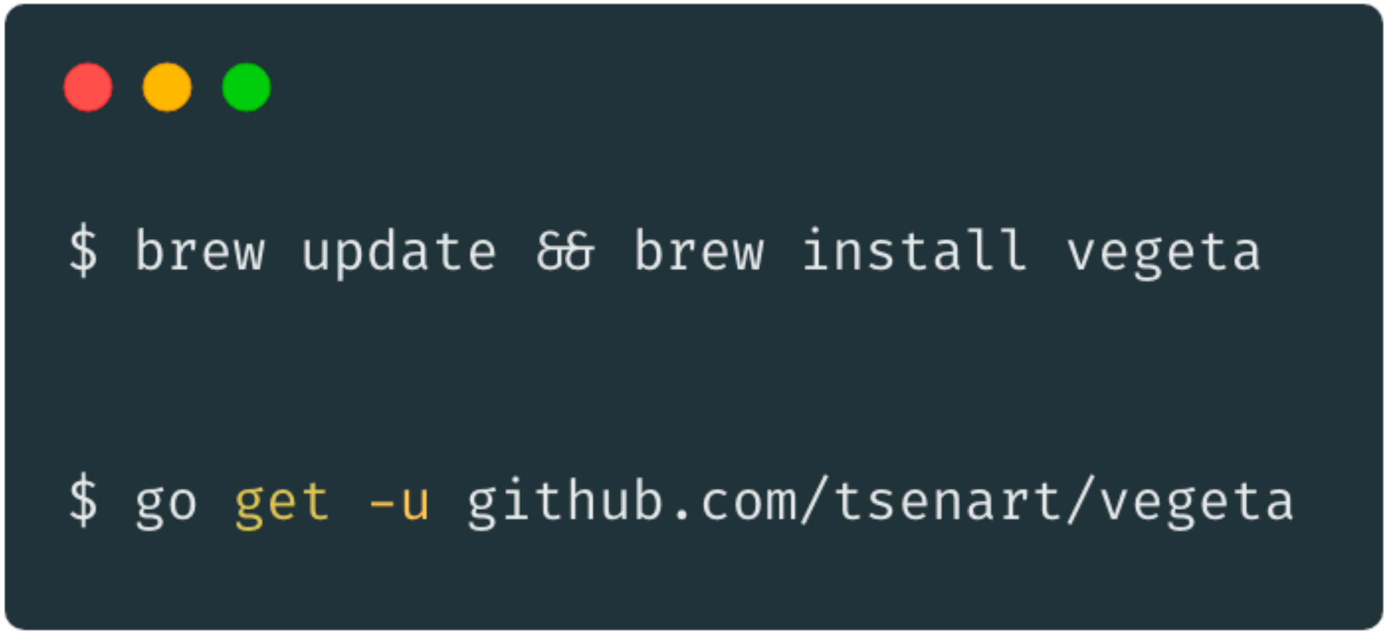
Vegeta

As vezes precisamos de um teste de esforço… **It's over 9000!**





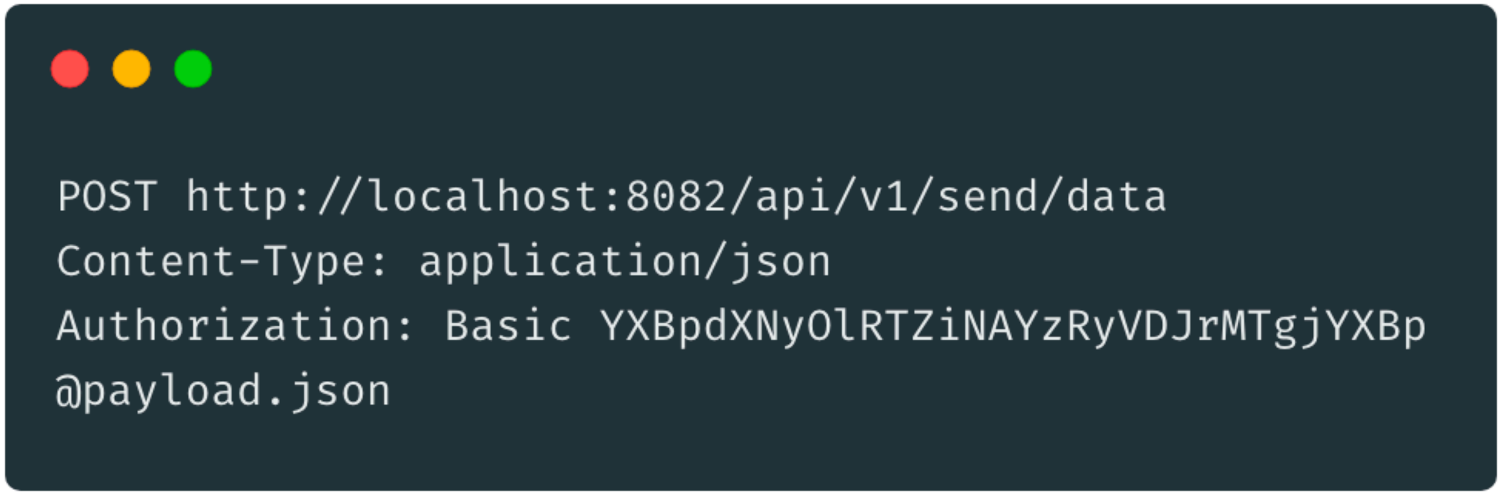
Instalando



```
$ brew update && brew install vegeta

$ go get -u github.com/tsenart/vegeta
```

github.com/tsenart/vegeta

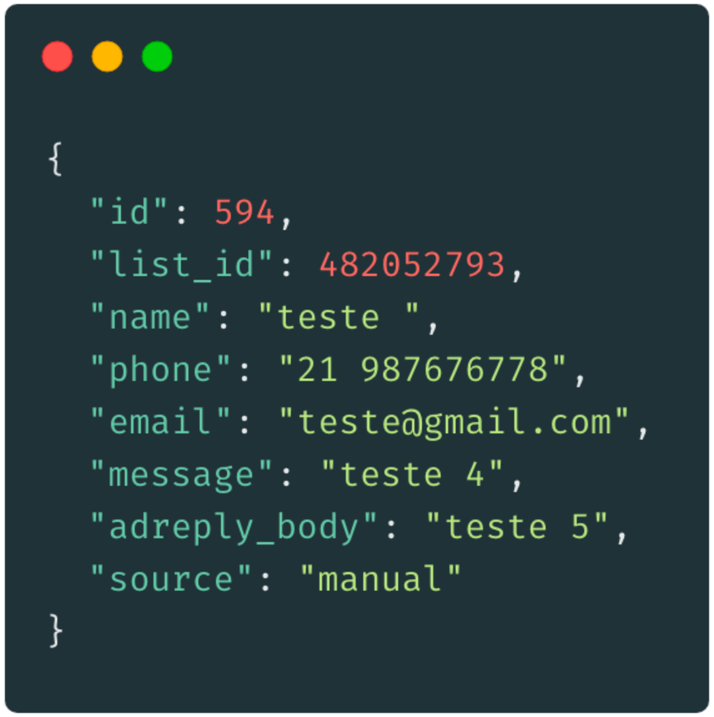
Preparando



```
POST http://localhost:8082/api/v1/send/data
Content-Type: application/json
Authorization: Basic YXBpdXNyOlRTZiNAYzRyVDJrMTgjYXBp
@payload.json
```

targets.txt

Preparando



```
{
  "id": 594,
  "list_id": 482052793,
  "name": "teste ",
  "phone": "21 987676778",
  "email": "teste@gmail.com",
  "message": "teste 4",
  "adreply_body": "teste 5",
  "source": "manual"
}
```

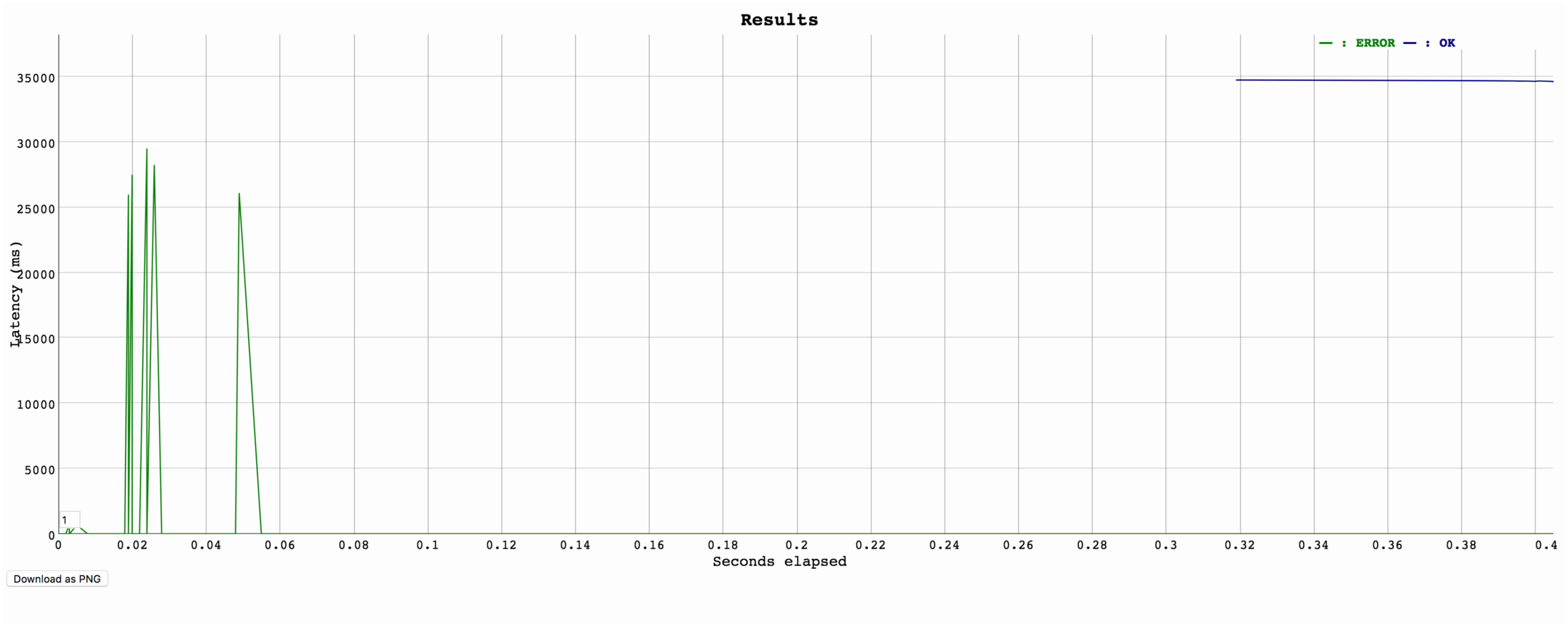
payload.json

Atacando

```
$ vegeta attack -duration=5s -rate=10000/1s -targets=targets.txt -output=results.bin


$ vegeta plot -title=Results results.bin > results-plot.html
```

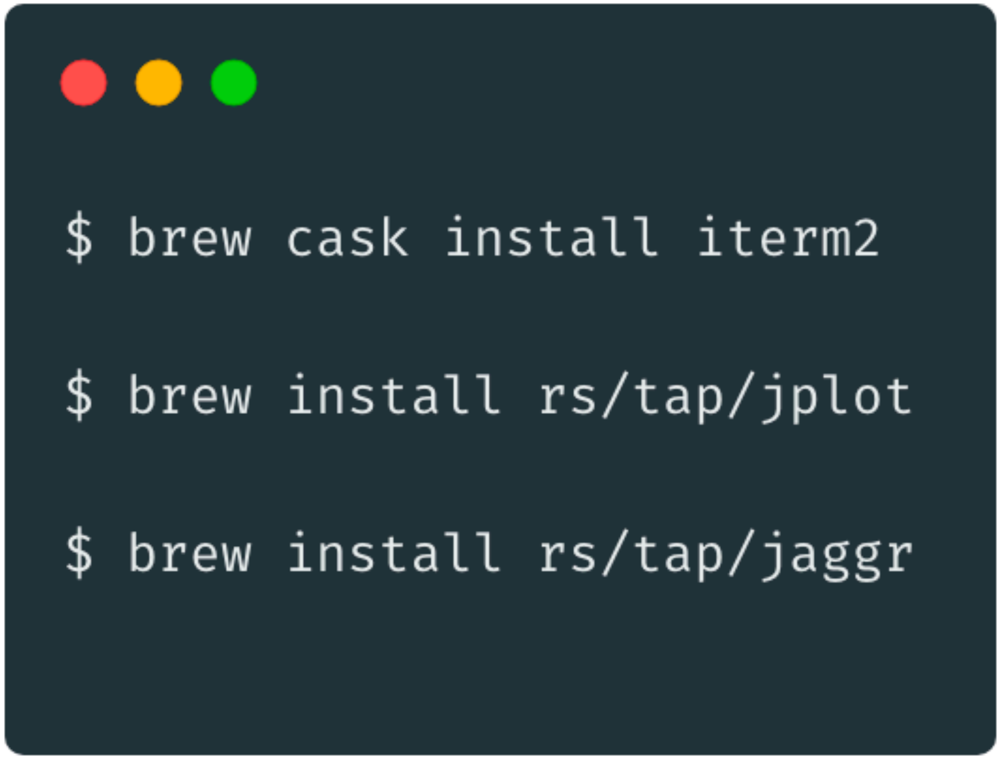
github.com/tsenart/vegeta

Analisando



Results
— : ERROR — : OK
Latency (ms)
0
5000
10000
15000
20000
25000
30000
35000
Seconds elapsed
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
0.22
0.24
0.26
0.28
0.3
0.32
0.34
0.36
0.38
0.4
1
Download as PNG

Vegeta + Jplot + Jaggr



```
$ brew cask install iterm2

$ brew install rs/tap/jplot

$ brew install rs/tap/jaggr
```

github.com/rs/jplot | github.com/rs/jaggr

Trocando isso...

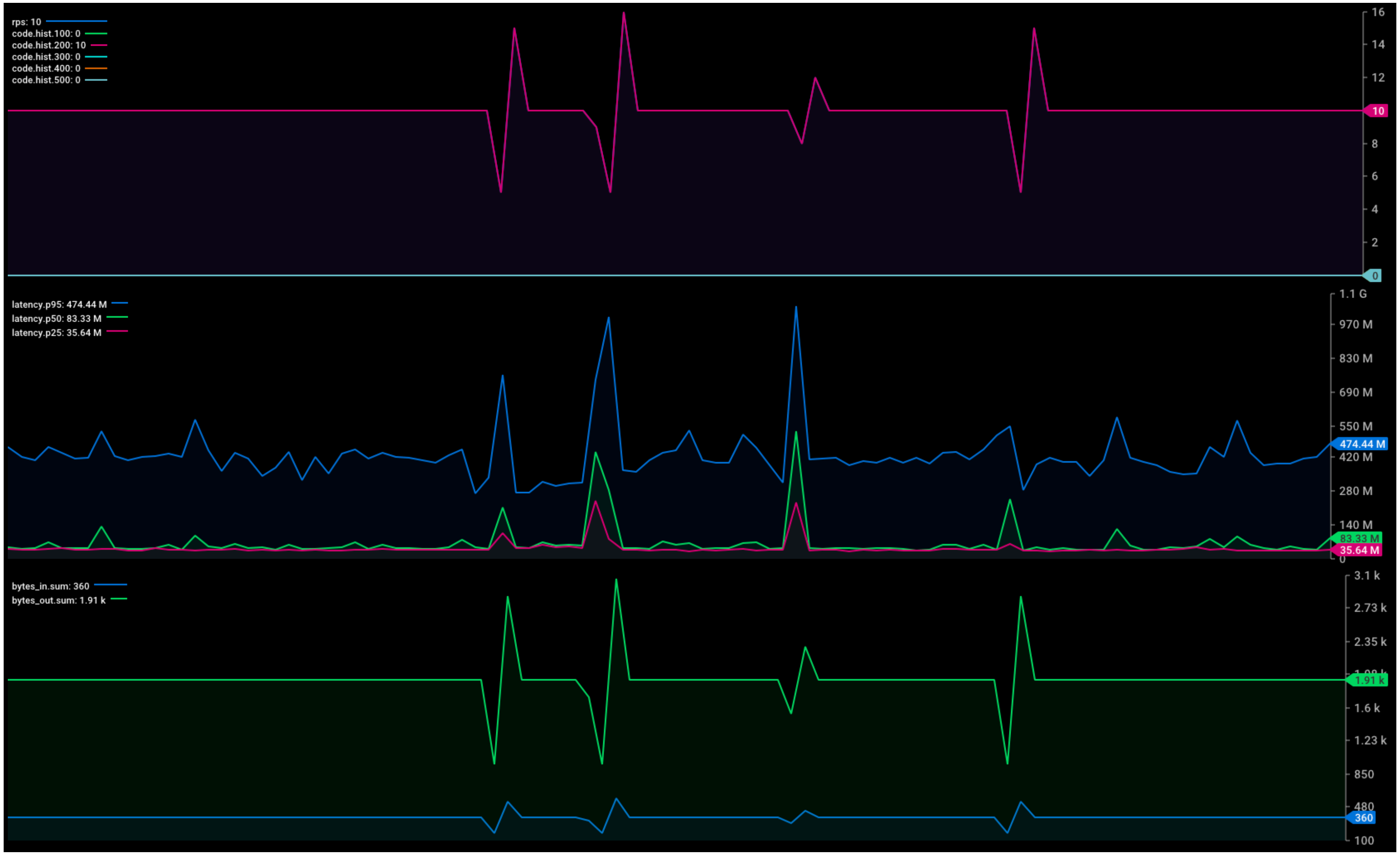
```
$ vegeta attack -duration=5s -rate=10000/1s -targets=targets.txt -output=results.bin


$ vegeta plot -title=Results results.bin > results-plot.html
```

Por isso...

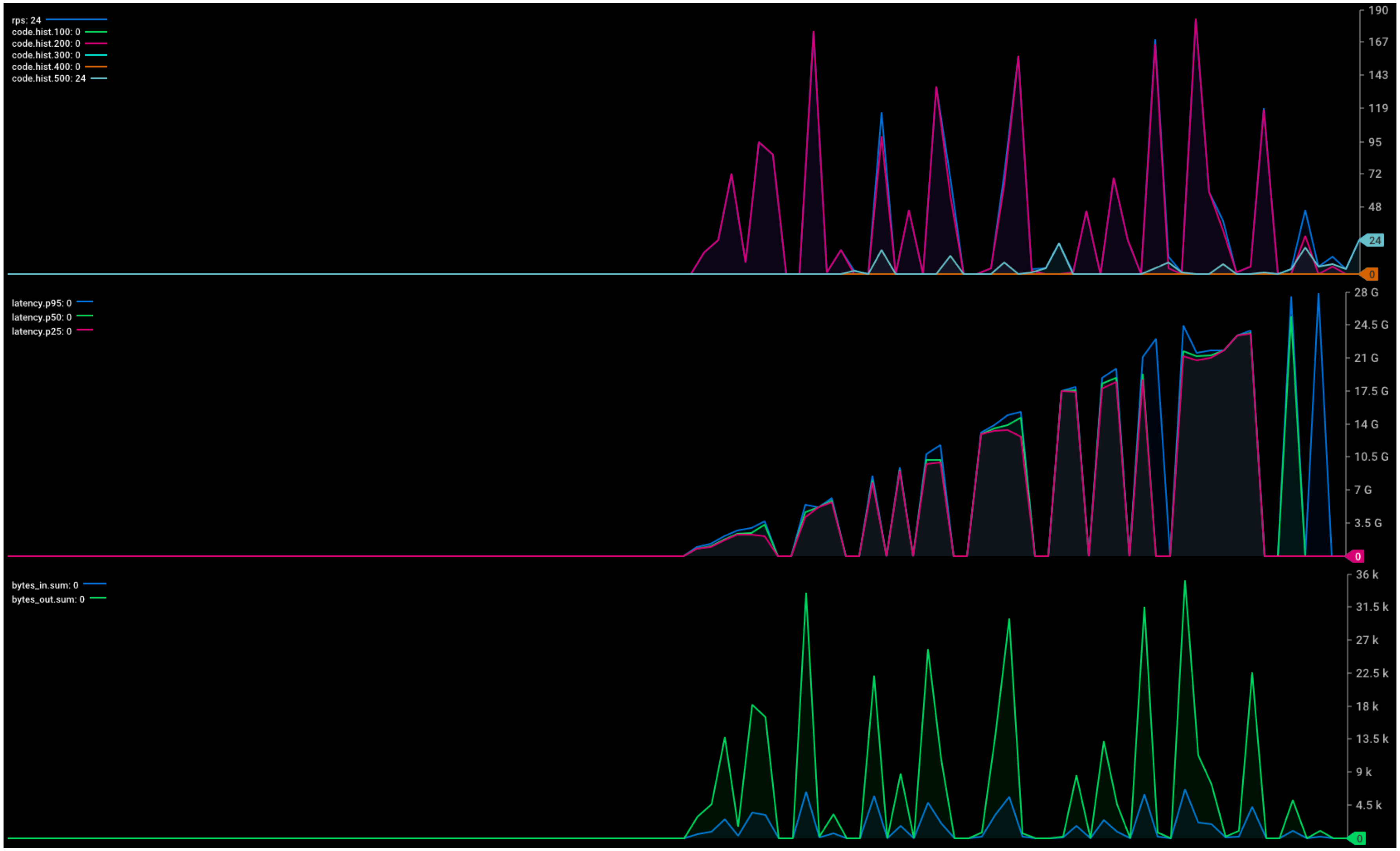
```
$ vegeta attack -rate 100 -duration 5m -targets=targets.txt | \
  vegeta encode | \
  jaggr @count=rps hist\[100,200,300,400,500\]:code p25,p50,p95:latency sum:bytes_in sum:bytes_out | \
  jplot rps+code.hist.100+code.hist.200+code.hist.300+code.hist.400+code.hist.500
latency.p95+latency.p50+latency.p25 bytes_in.sum+bytes_out.sum
```

github.com/tsenart/vegeta

10 req/sec

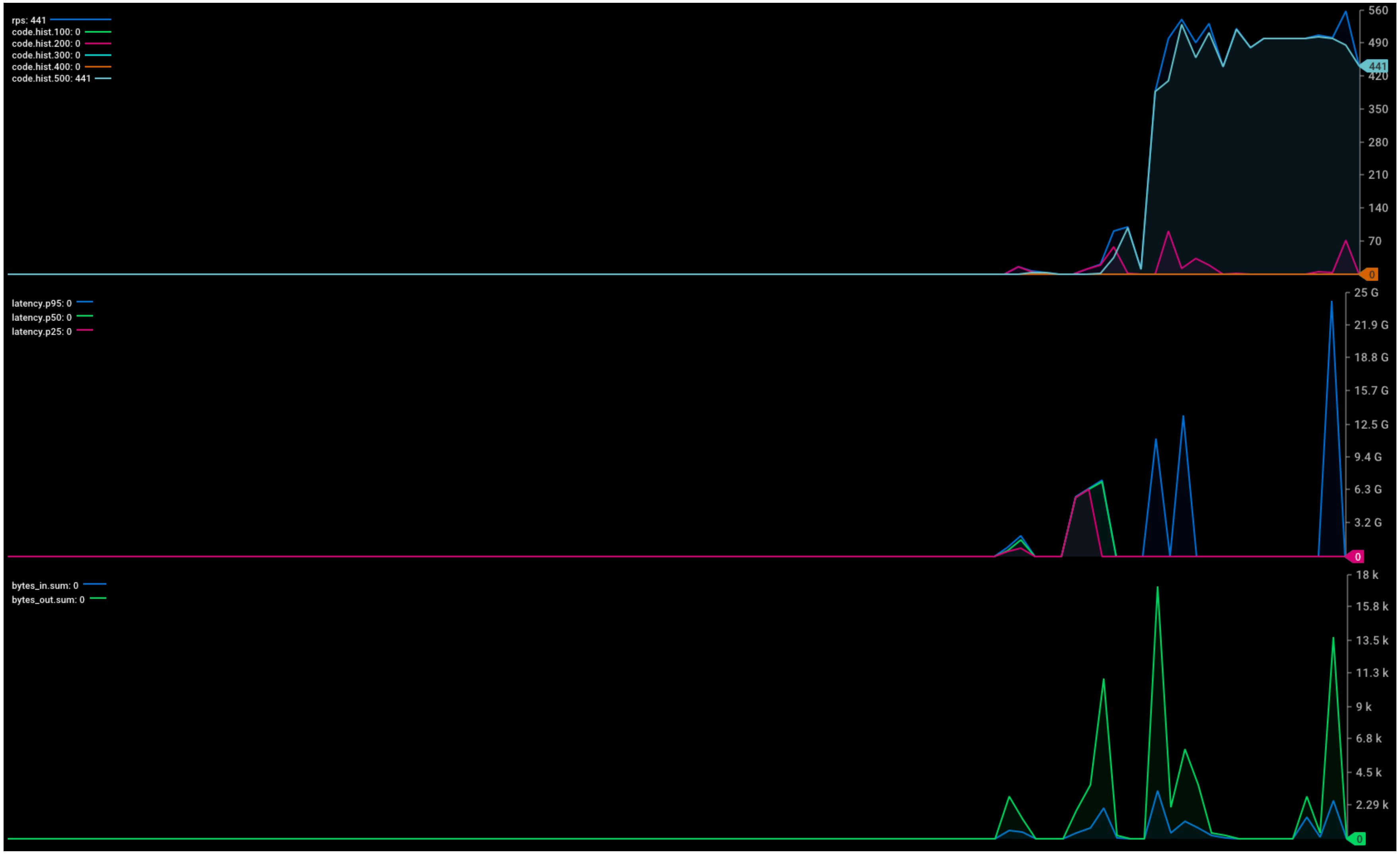

rps: 10
code.hist.100: 0
code.hist.200: 10
code.hist.300: 0
code.hist.400: 0
code.hist.500: 0
16
14
12
10
8
6
4
2
0
latency.p95: 474.44 M
latency.p50: 83.33 M
latency.p25: 35.64 M
1.1 G
970 M
830 M
690 M
550 M
474.44 M
420 M
280 M
140 M
83.33 M
35.64 M
0
bytes_in.sum: 360
bytes_out.sum: 1.91 k
3.1 k
2.73 k
2.35 k
1.91 k
1.6 k
1.23 k
850
480
360
100

100 req/sec



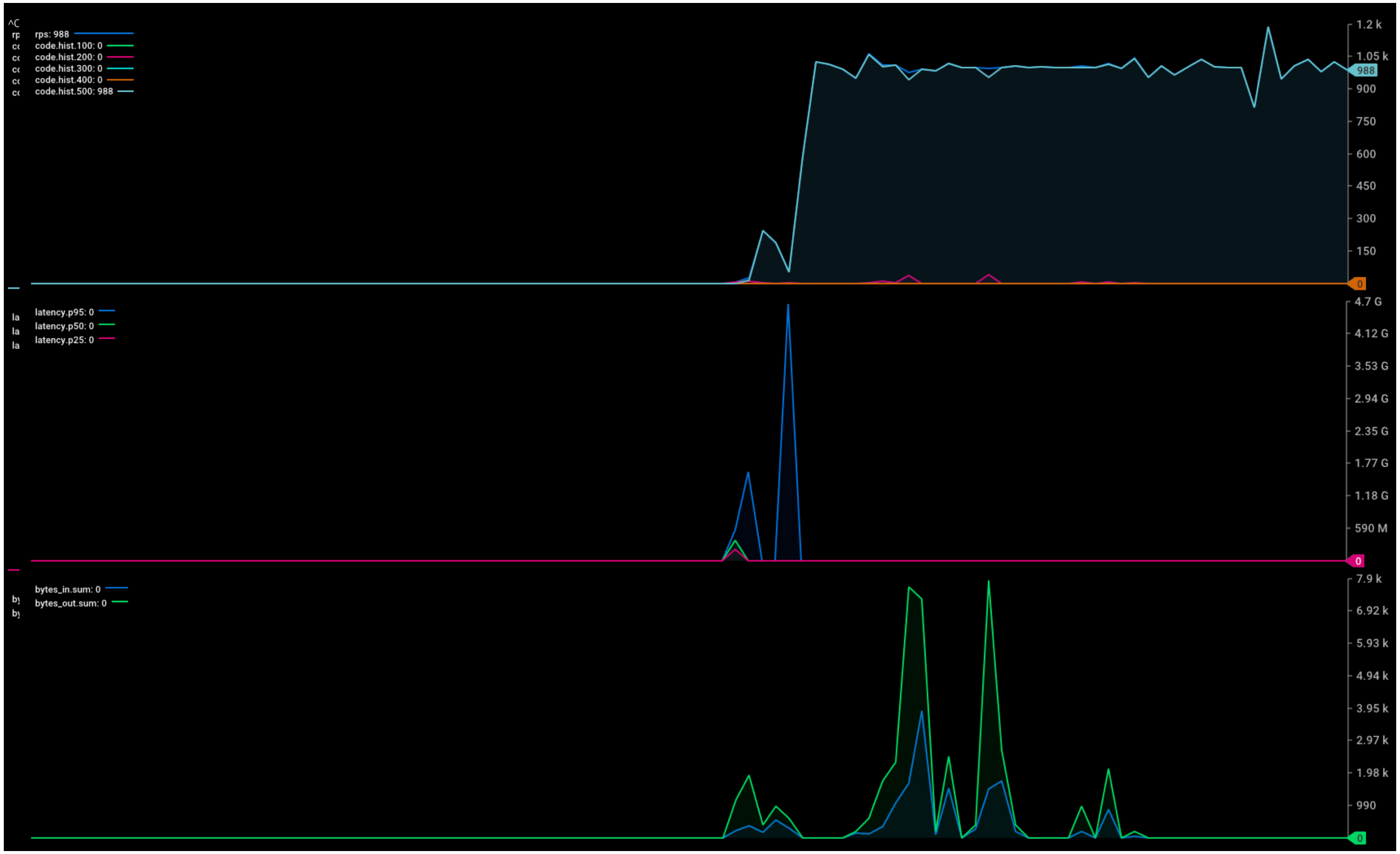
rps: 24
code.hist.100: 0
code.hist.200: 0
code.hist.300: 0
code.hist.400: 0
code.hist.500: 24
190
167
143
119
95
72
48
24
0
latency.p95: 0
latency.p50: 0
latency.p25: 0
28 G
24.5 G
21 G
17.5 G
14 G
10.5 G
7 G
3.5 G
0
bytes_in.sum: 0
bytes_out.sum: 0
36 k
31.5 k
27 k
22.5 k
18 k
13.5 k
9 k
4.5 k
0

500 req/sec

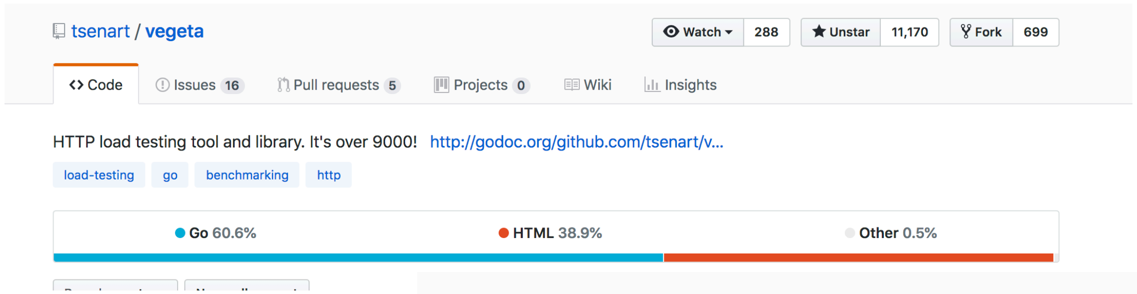


rps: 441
code.hist.100: 0
code.hist.200: 0
code.hist.300: 0
code.hist.400: 0
code.hist.500: 441
560
490
441
420
350
280
210
140
70
0
latency.p95: 0
latency.p50: 0
latency.p25: 0
25 G
21.9 G
18.8 G
15.7 G
12.5 G
9.4 G
6.3 G
3.2 G
0
bytes_in.sum: 0
bytes_out.sum: 0
18 k
15.8 k
13.5 k
11.3 k
9 k
6.8 k
4.5 k
2.29 k
0

1000 req/sec


^C
rps: 988
code.hist.100: 0
code.hist.200: 0
code.hist.300: 0
code.hist.400: 0
code.hist.500: 988
1.2 k
1.05 k
988
900
750
600
450
300
150
0
latency.p95: 0
latency.p50: 0
latency.p25: 0
4.7 G
4.12 G
3.53 G
2.94 G
2.35 G
1.77 G
1.18 G
590 M
0
bytes_in.sum: 0
bytes_out.sum: 0
7.9 k
6.92 k
5.93 k
4.94 k
3.95 k
2.97 k
1.98 k
990
0

vegeta | jplot | jaggr



tsenart / vegeta

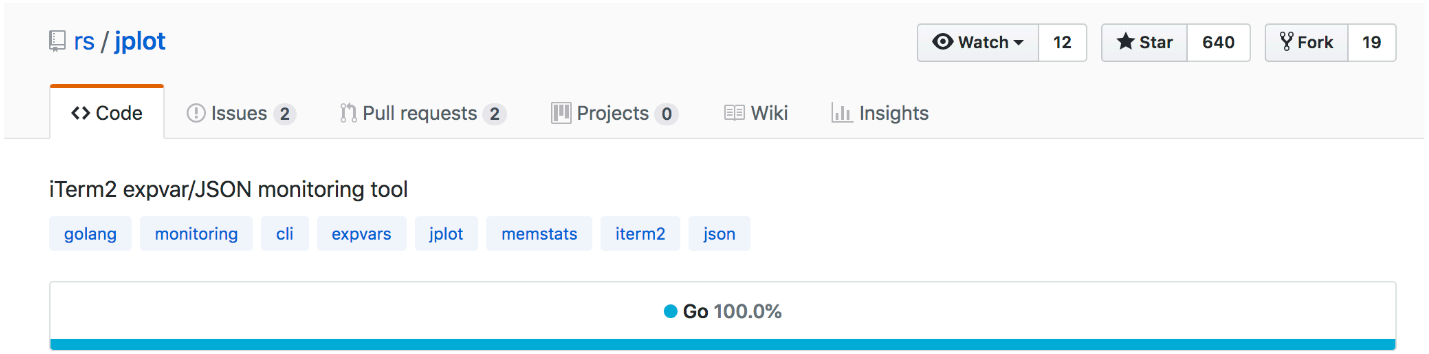
Watch 288 | Unstar 11,170 | Fork 699

Code | Issues 16 | Pull requests 5 | Projects 0 | Wiki | Insights

HTTP load testing tool and library. It's over 9000! http://godoc.org/github.com/tsenart/v...

load-testing | go | benchmarking | http

● Go 60.6% ● HTML 38.9% ● Other 0.5%



rs / jplot

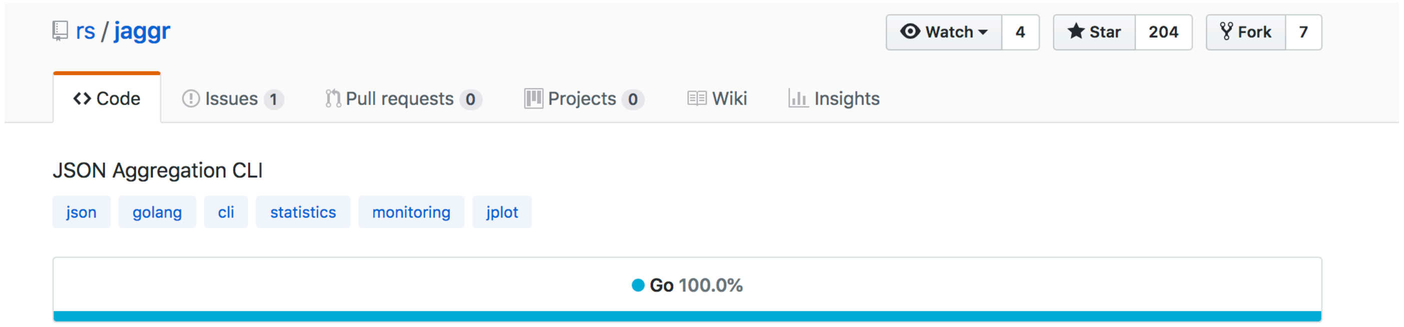
Watch 12 | Star 640 | Fork 19

Code | Issues 2 | Pull requests 2 | Projects 0 | Wiki | Insights

iTerm2 expvar/JSON monitoring tool

golang | monitoring | cli | expvars | jplot | memstats | iterm2 | json

● Go 100.0%



rs / jaggr

Watch 4 | Star 204 | Fork 7

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Insights

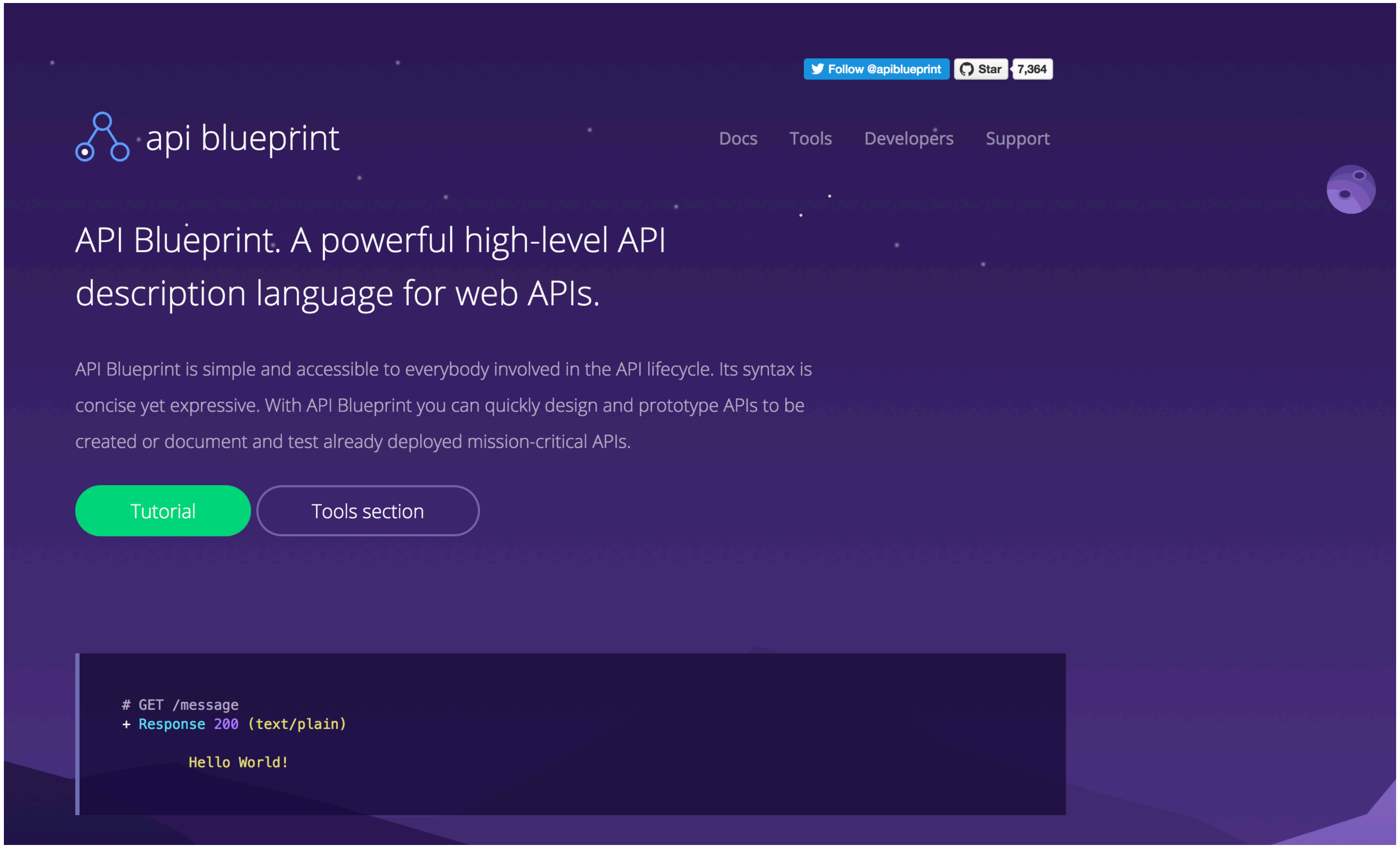
JSON Aggregation CLI

json | golang | cli | statistics | monitoring | jplot

● Go 100.0%

Todos escritos em Go

Informação adicional… Mocks…



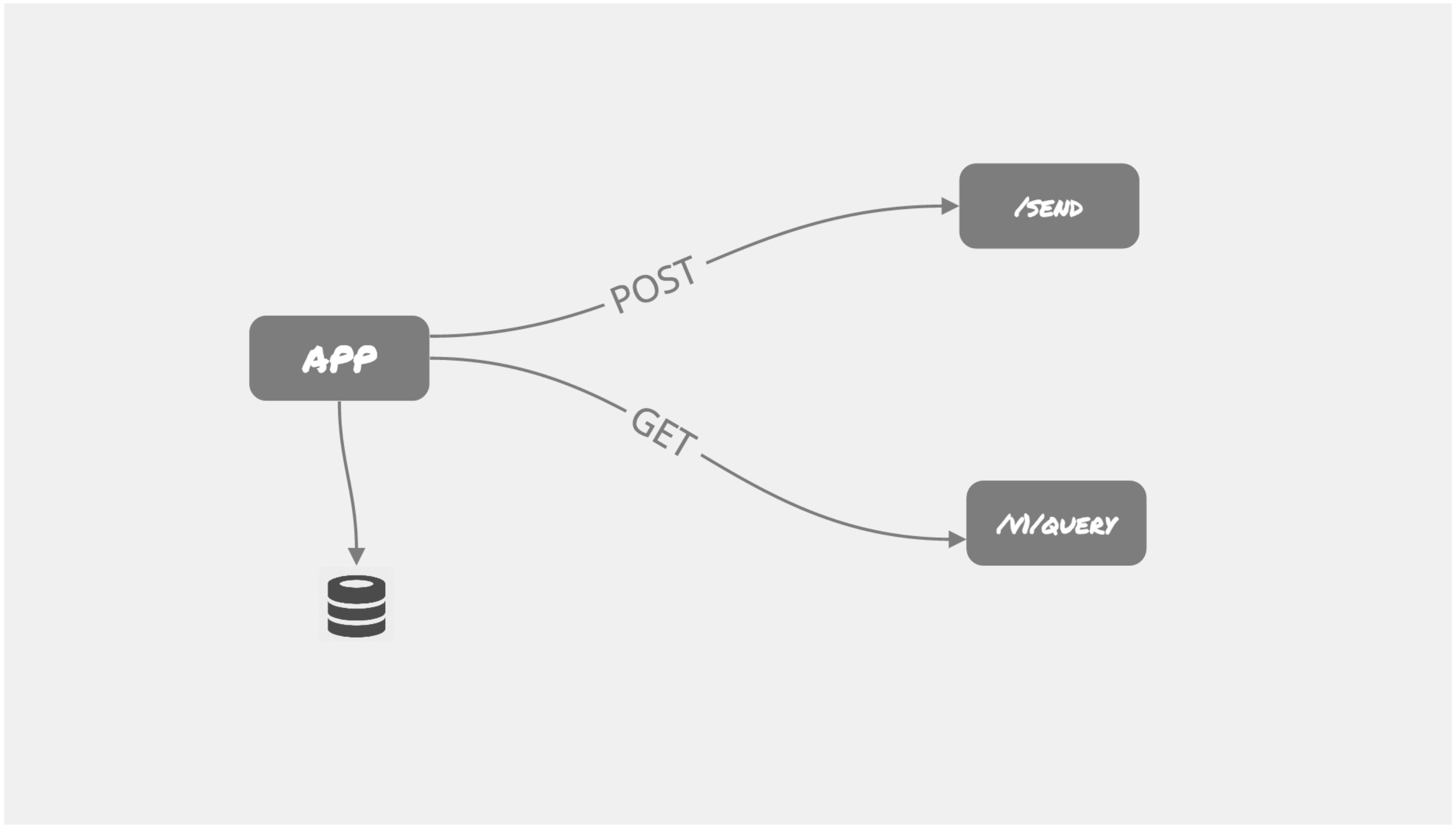
API Blueprint. A powerful high-level API description language for web APIs.

API Blueprint is simple and accessible to everybody involved in the API lifecycle. Its syntax is concise yet expressive. With API Blueprint you can quickly design and prototype APIs to be created or document and test already deployed mission-critical APIs.

```
# GET /message
+ Response 200 (text/plain)

        Hello World!
```

apiblueprint.org

Informação adicional… Mocks…



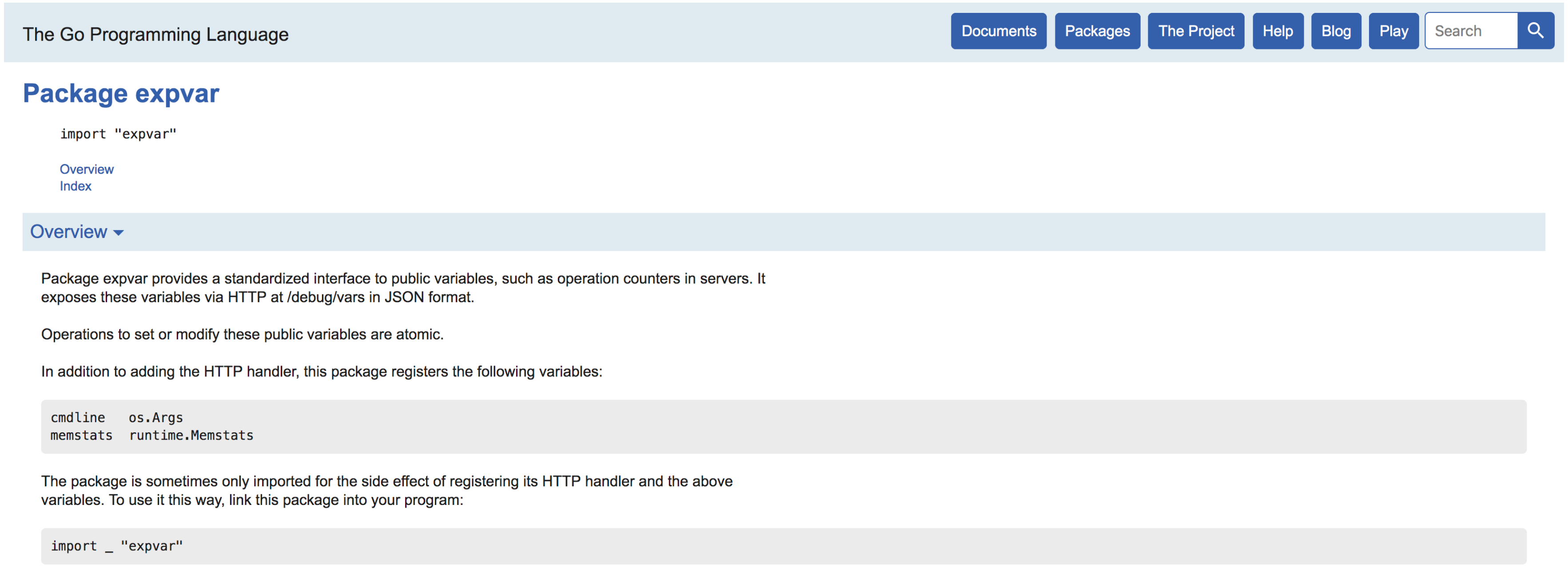
Regra de ouro #2

Não subestime a quantidade de requisições

Faça um eletrocardiograma

Olhe o problema de perto

expvar



Package expvar

```
import "expvar"
```

Overview
Index

Overview ▾

Package expvar provides a standardized interface to public variables, such as operation counters in servers. It exposes these variables via HTTP at /debug/vars in JSON format.

Operations to set or modify these public variables are atomic.

In addition to adding the HTTP handler, this package registers the following variables:

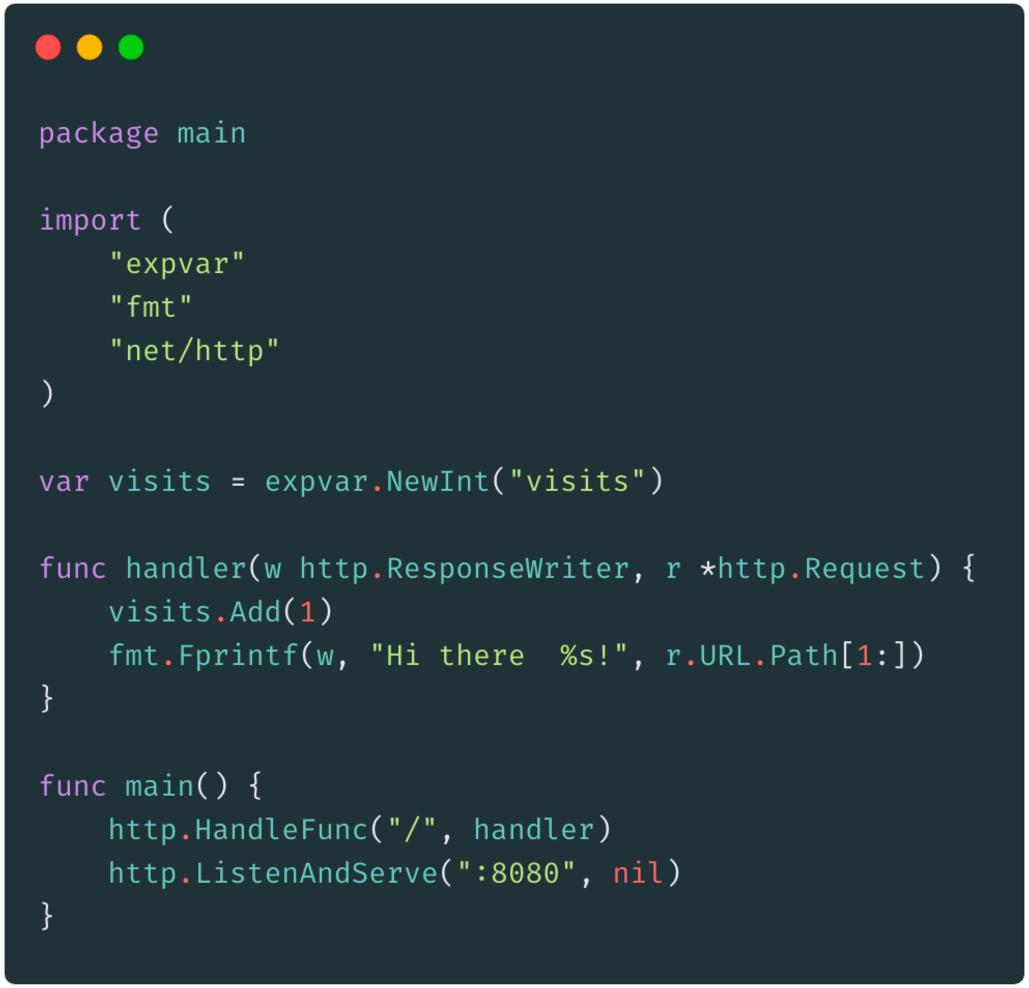
```
cmdline   os.Args
memstats  runtime.Memstats
```

The package is sometimes only imported for the side effect of registering its HTTP handler and the above variables. To use it this way, link this package into your program:

```
import _ "expvar"
```

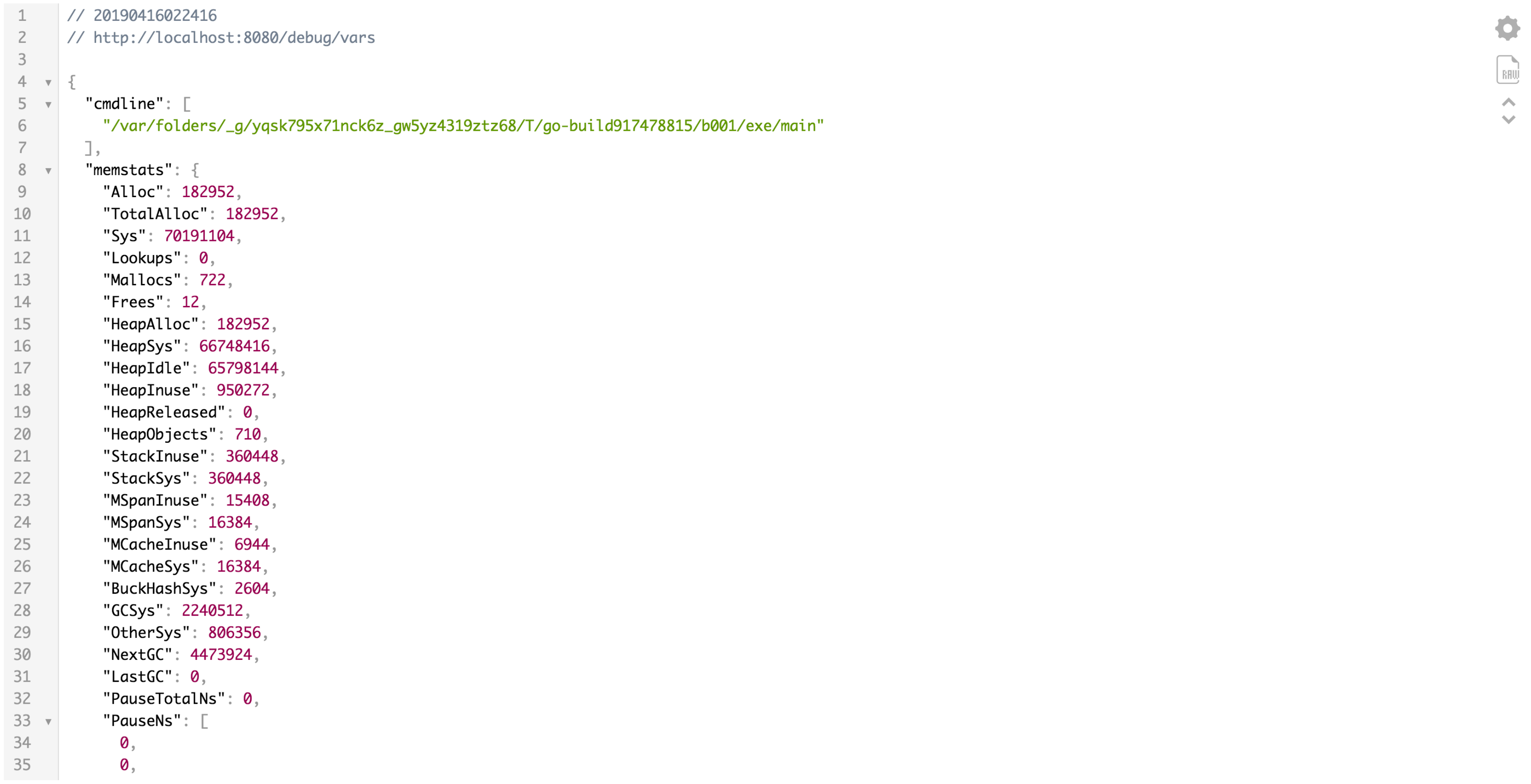
golang.org/pkg/expvar/

expvar



```go
package main

import (
    "expvar"
    "fmt"
    "net/http"
)

var visits = expvar.NewInt("visits")

func handler(w http.ResponseWriter, r *http.Request) {
    visits.Add(1)
    fmt.Fprintf(w, "Hi there  %s!", r.URL.Path[1:])
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

expvar

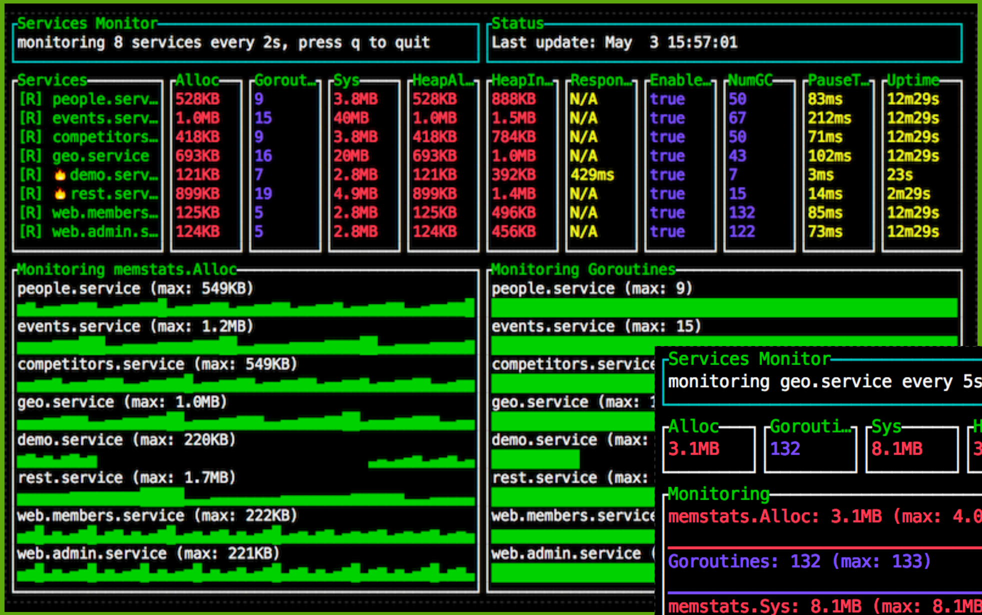


```
// 20190416022416
// http://localhost:8080/debug/vars

{
  "cmdline": [
    "/var/folders/_g/yqsk795x71nck6z_gw5yz4319ztz68/T/go-build917478815/b001/exe/main"
  ],
  "memstats": {
    "Alloc": 182952,
    "TotalAlloc": 182952,
    "Sys": 70191104,
    "Lookups": 0,
    "Mallocs": 722,
    "Frees": 12,
    "HeapAlloc": 182952,
    "HeapSys": 66748416,
    "HeapIdle": 65798144,
    "HeapInuse": 950272,
    "HeapReleased": 0,
    "HeapObjects": 710,
    "StackInuse": 360448,
    "StackSys": 360448,
    "MSpanInuse": 15408,
    "MSpanSys": 16384,
    "MCacheInuse": 6944,
    "MCacheSys": 16384,
    "BuckHashSys": 2604,
    "GCSys": 2240512,
    "OtherSys": 806356,
    "NextGC": 4473924,
    "LastGC": 0,
    "PauseTotalNs": 0,
    "PauseNs": [
      0,
      0,
```

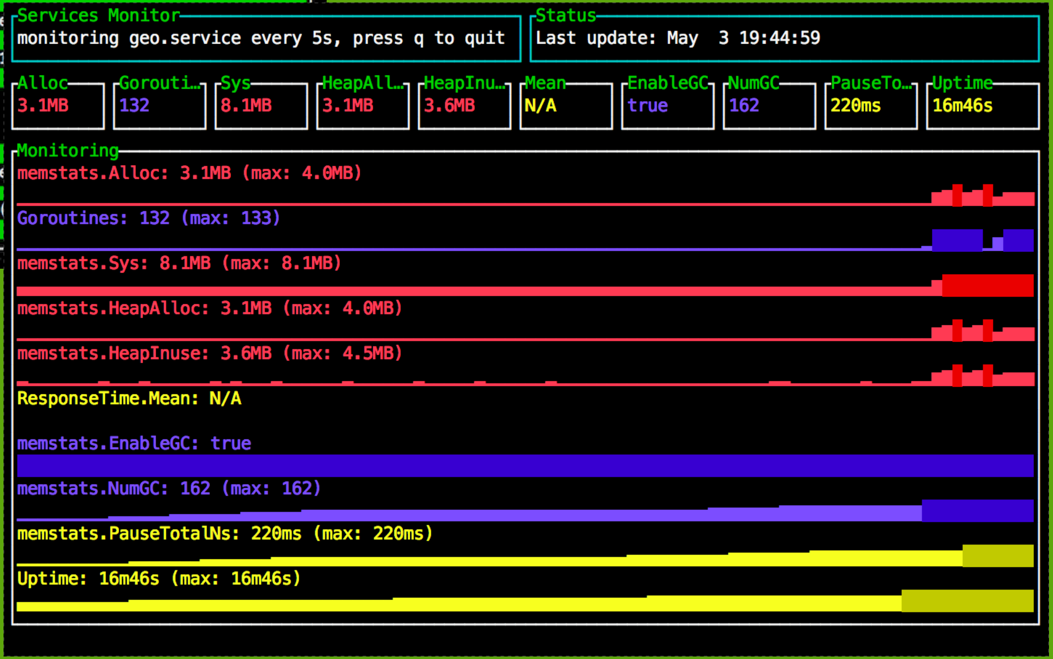
http://localhost:8080/debug/vars

ExpvarMon

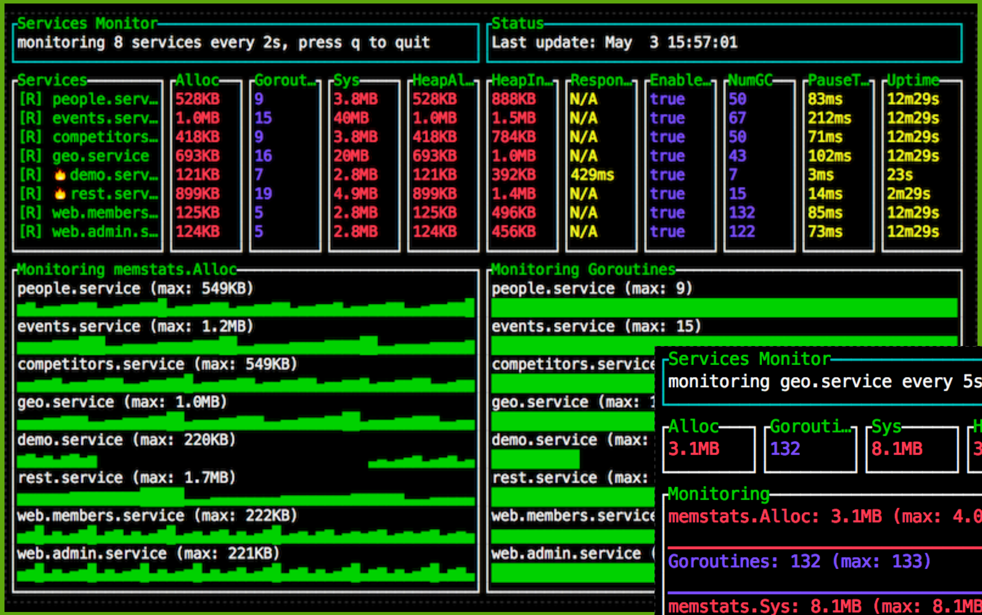
Monitore suas aplicações Go bem de perto



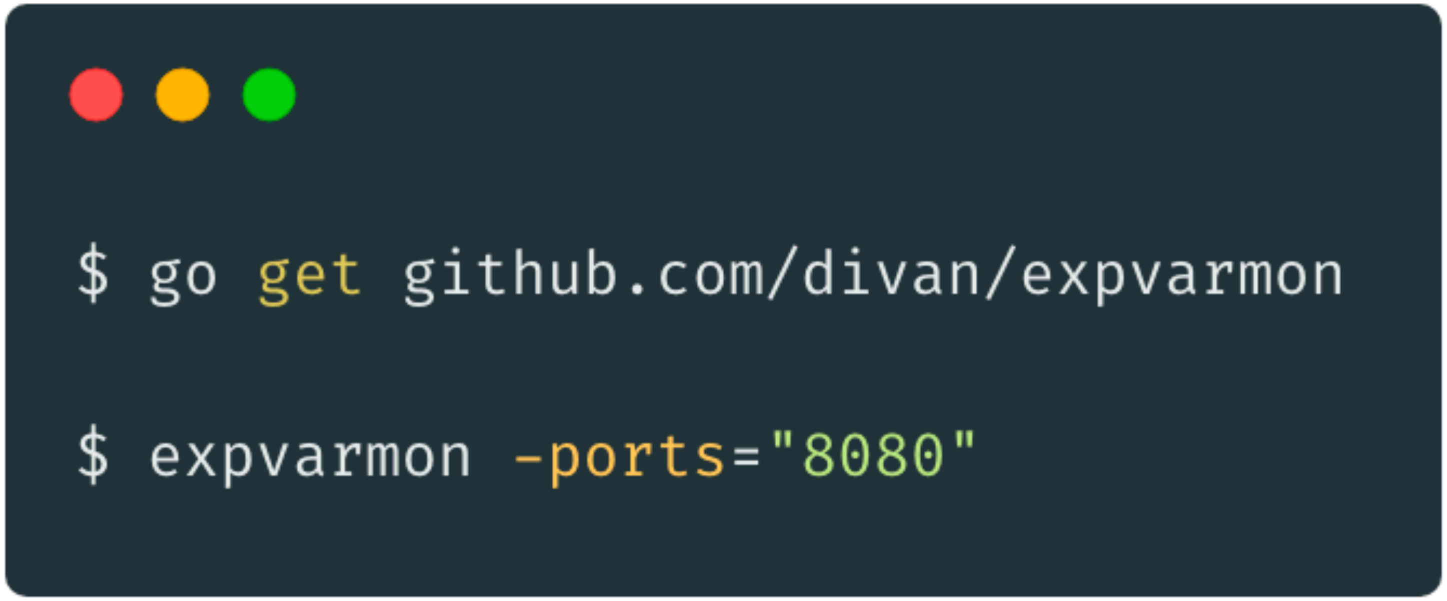
```
Services Monitor                                        Status
monitoring 8 services every 2s, press q to quit         Last update: May  3 15:57:01

Services            Alloc   Gorout… Sys    HeapAl… HeapIn… Respon… Enable… NumGC  PauseT… Uptime
[R] people.serv…    528KB   9       3.8MB  528KB   888KB   N/A     true    50     83ms    12m29s
[R] events.serv…    1.0MB   15      40MB   1.0MB   1.5MB   N/A     true    67     212ms   12m29s
[R] competitors…    418KB   9       3.8MB  418KB   784KB   N/A     true    50     71ms    12m29s
[R] geo.service     693KB   16      20MB   693KB   1.0MB   N/A     true    43     102ms   12m29s
[R] 🔥demo.serv…    121KB   7       2.8MB  121KB   392KB   429ms   true    7      3ms     23s
[R] 🔥rest.serv…    899KB   19      4.9MB  899KB   1.4MB   N/A     true    15     14ms    2m29s
[R] web.members…    125KB   5       2.8MB  125KB   496KB   N/A     true    132    85ms    12m29s
[R] web.admin.s…    124KB   5       2.8MB  124KB   456KB   N/A     true    122    73ms    12m29s

Monitoring memstats.Alloc                               Monitoring Goroutines
people.service (max: 549KB)                             people.service (max: 9)
events.service (max: 1.2MB)                             events.service (max: 15)
competitors.service (max: 549KB)                        competitors.service
geo.service (max: 1.0MB)                                geo.service (max: 1
demo.service (max: 220KB)                               demo.service (max:
rest.service (max: 1.7MB)                               rest.service (max:
web.members.service (max: 222KB)                        web.members.service
web.admin.service (max: 221KB)                          web.admin.service (
```



```
Services Monitor                                        Status
monitoring geo.service every 5s, press q to quit        Last update: May  3 19:44:59

Alloc    Libraries…  Sys     HeapAll…  HeapInu…  Mean   EnableGC  NumGC  PauseTo…  Uptime
3.1MB    132         8.1MB   3.1MB     3.6MB     N/A    true      162    220ms     16m46s

Monitoring
memstats.Alloc: 3.1MB (max: 4.0MB)
Goroutines: 132 (max: 133)
memstats.Sys: 8.1MB (max: 8.1MB)
memstats.HeapAlloc: 3.1MB (max: 4.0MB)
memstats.HeapInuse: 3.6MB (max: 4.5MB)
ResponseTime.Mean: N/A
memstats.EnableGC: true
memstats.NumGC: 162 (max: 162)
memstats.PauseTotalNs: 220ms (max: 220ms)
Uptime: 16m46s (max: 16m46s)
```



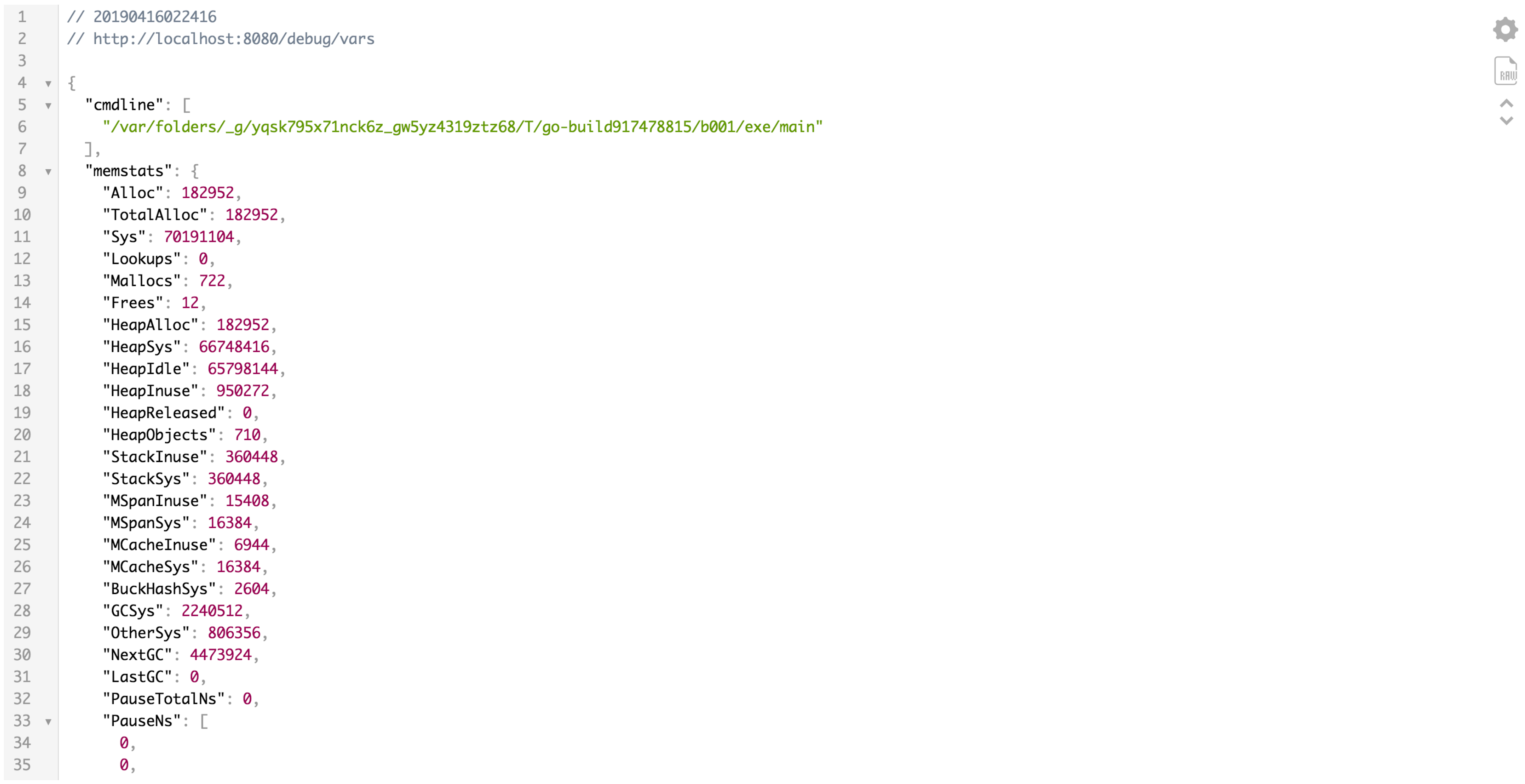
Instalando e executando



```
$ go get github.com/divan/expvarmon

$ expvarmon -ports="8080"
```

github.com/divan/expvarmon

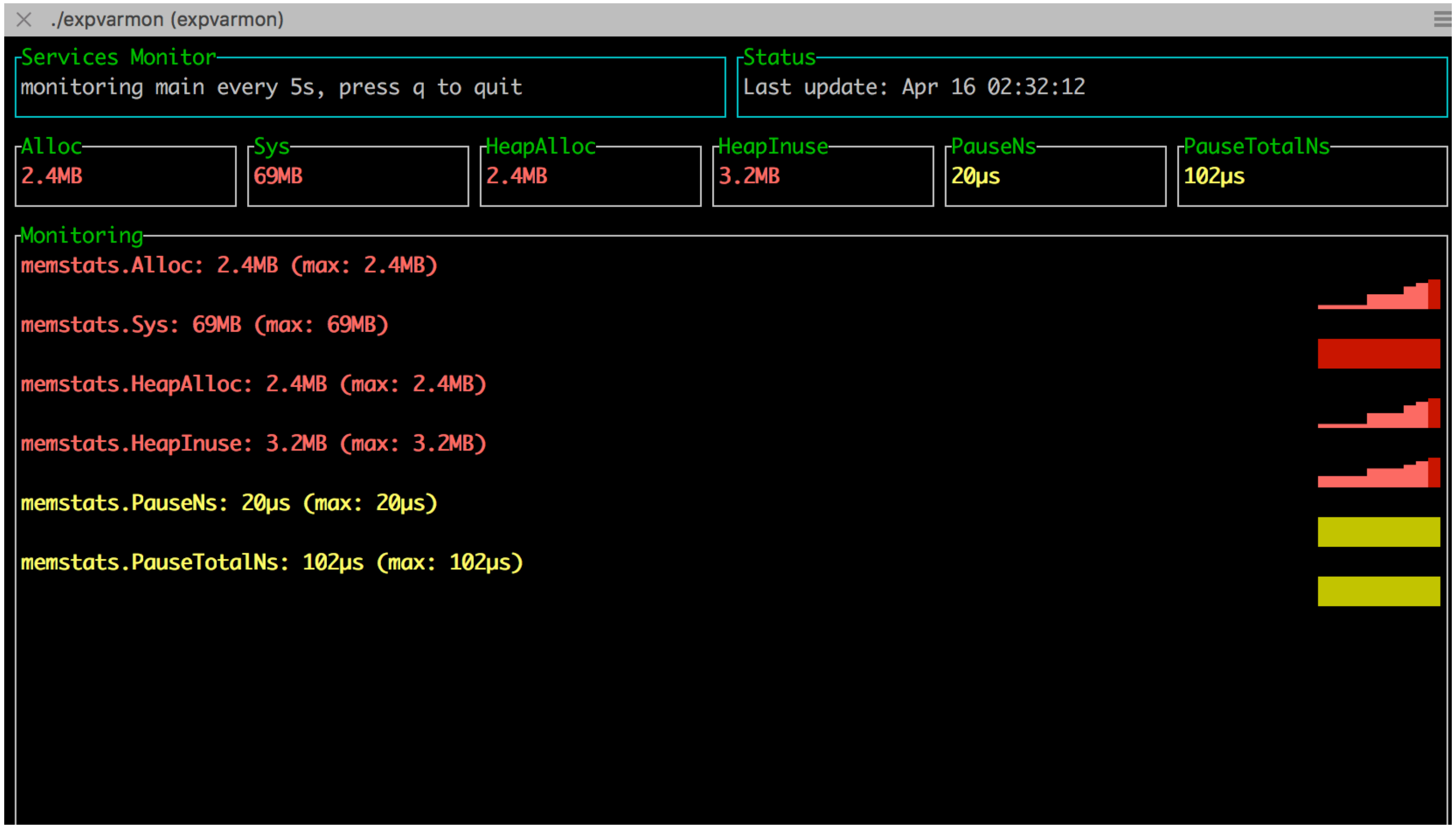
expvar



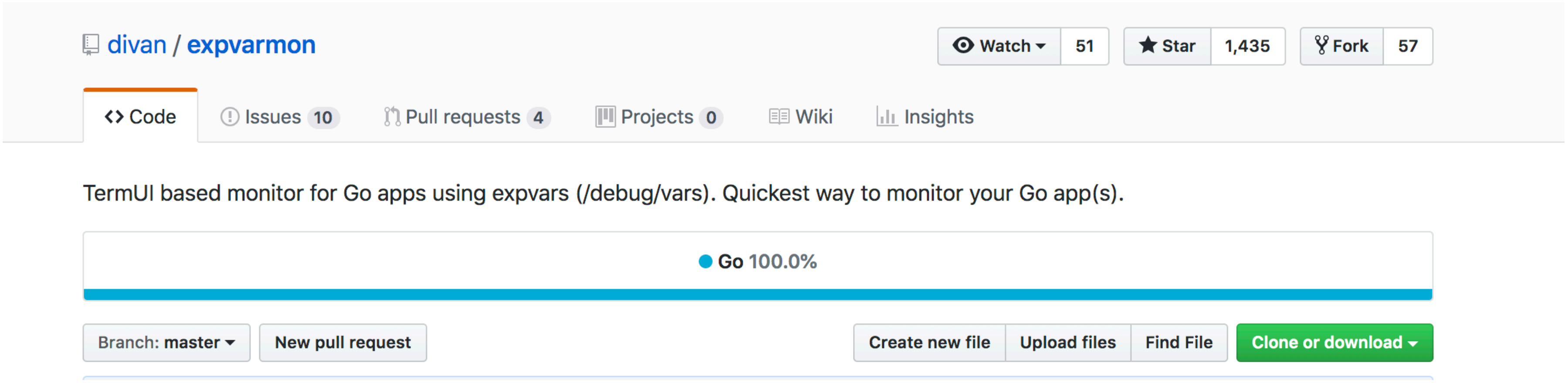
```
// 20190416022416
// http://localhost:8080/debug/vars

{
  "cmdline": [
    "/var/folders/_g/yqsk795x71nck6z_gw5yz4319ztz68/T/go-build917478815/b001/exe/main"
  ],
  "memstats": {
    "Alloc": 182952,
    "TotalAlloc": 182952,
    "Sys": 70191104,
    "Lookups": 0,
    "Mallocs": 722,
    "Frees": 12,
    "HeapAlloc": 182952,
    "HeapSys": 66748416,
    "HeapIdle": 65798144,
    "HeapInuse": 950272,
    "HeapReleased": 0,
    "HeapObjects": 710,
    "StackInuse": 360448,
    "StackSys": 360448,
    "MSpanInuse": 15408,
    "MSpanSys": 16384,
    "MCacheInuse": 6944,
    "MCacheSys": 16384,
    "BuckHashSys": 2604,
    "GCSys": 2240512,
    "OtherSys": 806356,
    "NextGC": 4473924,
    "LastGC": 0,
    "PauseTotalNs": 0,
    "PauseNs": [
      0,
      0,
```

http://localhost:8080/debug/vars

50 rps


```
./expvarmon (expvarmon)

Services Monitor                                    Status
monitoring main every 5s, press q to quit           Last update: Apr 16 02:32:12

Alloc        Sys          HeapAlloc    HeapInuse    PauseNs      PauseTotalNs
2.4MB        69MB         2.4MB        3.2MB        20µs         102µs

Monitoring
memstats.Alloc: 2.4MB (max: 2.4MB)

memstats.Sys: 69MB (max: 69MB)

memstats.HeapAlloc: 2.4MB (max: 2.4MB)

memstats.HeapInuse: 3.2MB (max: 3.2MB)

memstats.PauseNs: 20µs (max: 20µs)

memstats.PauseTotalNs: 102µs (max: 102µs)
```

Escrito em Go


divan / expvarmon

Watch 51 | Star 1,435 | Fork 57

Code | Issues 10 | Pull requests 4 | Projects 0 | Wiki | Insights

TermUI based monitor for Go apps using expvars (/debug/vars). Quickest way to monitor your Go app(s).

● Go 100.0%

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

Faça um checkup geral

Já vimos o micro, mas não esqueça do macro

Reporting metrics (PUSH) vs Collecting metrics (PULL)

PUSH

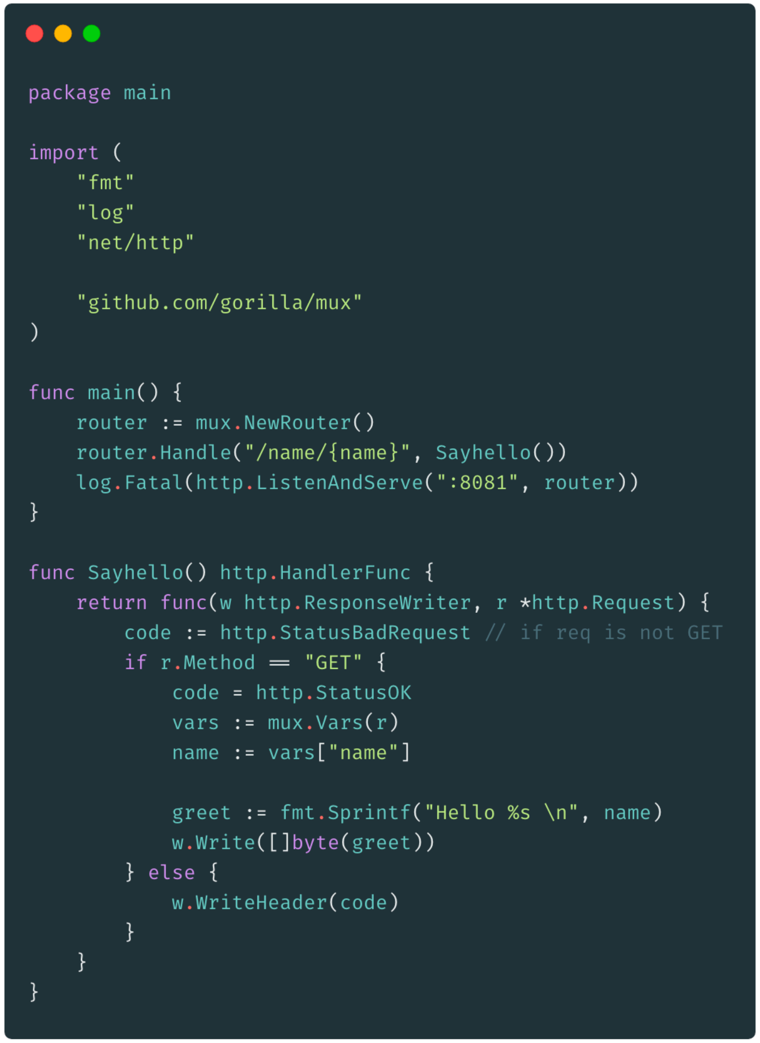
Depois de coletas as métricas, ele envia as informações medidas para o serviço.

PULL

Você expõe suas informações em um Endpoint definido

Prometheus

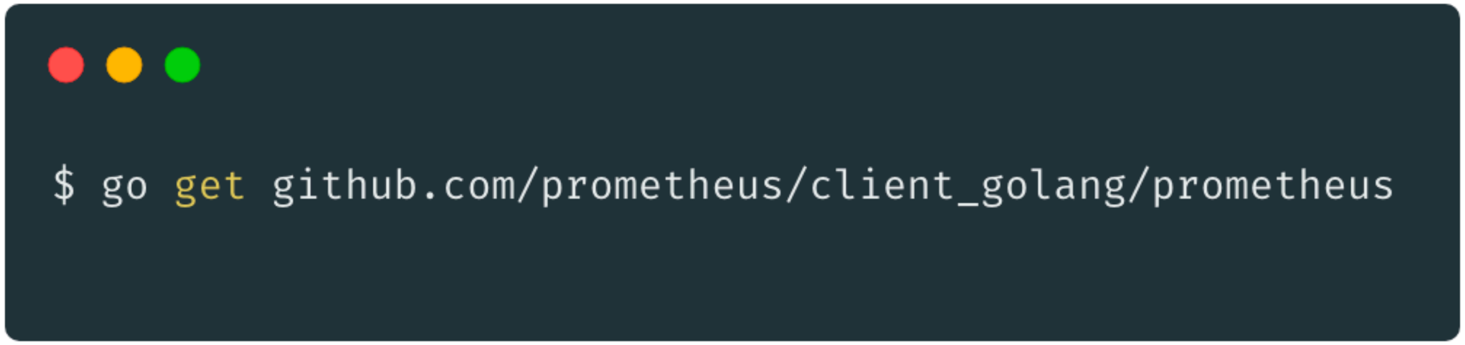
Começando pelo básico



```go
package main

import (
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

func main() {
    router := mux.NewRouter()
    router.Handle("/name/{name}", Sayhello())
    log.Fatal(http.ListenAndServe(":8081", router))
}

func Sayhello() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        code := http.StatusBadRequest // if req is not GET
        if r.Method == "GET" {
            code = http.StatusOK
            vars := mux.Vars(r)
            name := vars["name"]

            greet := fmt.Sprintf("Hello %s \n", name)
            w.Write([]byte(greet))
        } else {
            w.WriteHeader(code)
        }
    }
}
```

medium.com/@gsisimogang/instrumenting-golang-server-in-5-min-c1c32489add3

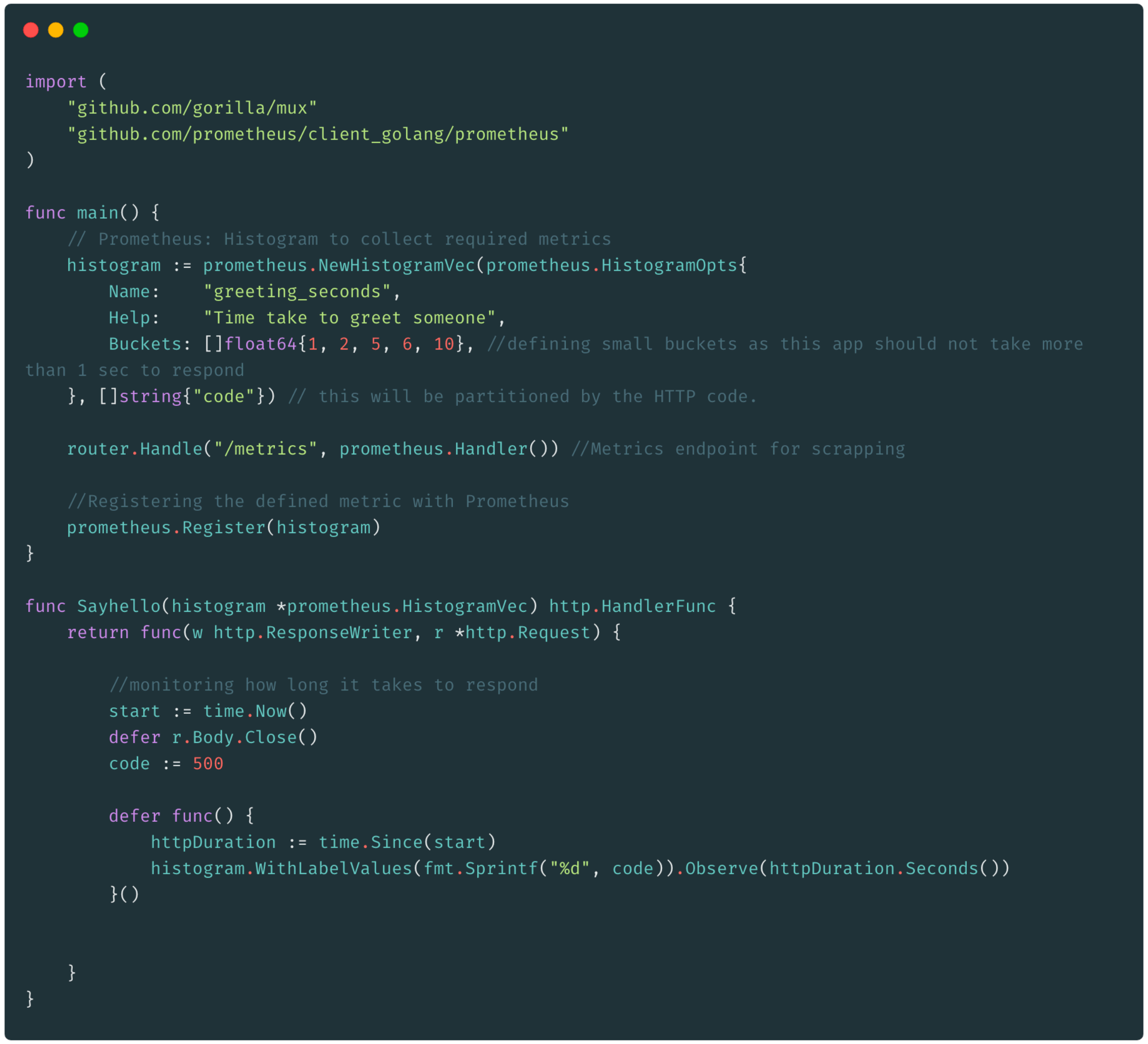
Adicionando dependencia



```
$ go get github.com/prometheus/client_golang/prometheus
```

medium.com/@gsisimogang/instrumenting-golang-server-in-5-min-c1c32489add3

Configurando



```go
import (
    "github.com/gorilla/mux"
    "github.com/prometheus/client_golang/prometheus"
)

func main() {
    // Prometheus: Histogram to collect required metrics
    histogram := prometheus.NewHistogramVec(prometheus.HistogramOpts{
        Name:    "greeting_seconds",
        Help:    "Time take to greet someone",
        Buckets: []float64{1, 2, 5, 6, 10}, //defining small buckets as this app should not take more
than 1 sec to respond
    }, []string{"code"}) // this will be partitioned by the HTTP code.

    router.Handle("/metrics", prometheus.Handler()) //Metrics endpoint for scrapping

    //Registering the defined metric with Prometheus
    prometheus.Register(histogram)
}

func Sayhello(histogram *prometheus.HistogramVec) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {

        //monitoring how long it takes to respond
        start := time.Now()
        defer r.Body.Close()
        code := 500

        defer func() {
            httpDuration := time.Since(start)
            histogram.WithLabelValues(fmt.Sprintf("%d", code)).Observe(httpDuration.Seconds())
        }()


    }
}
```

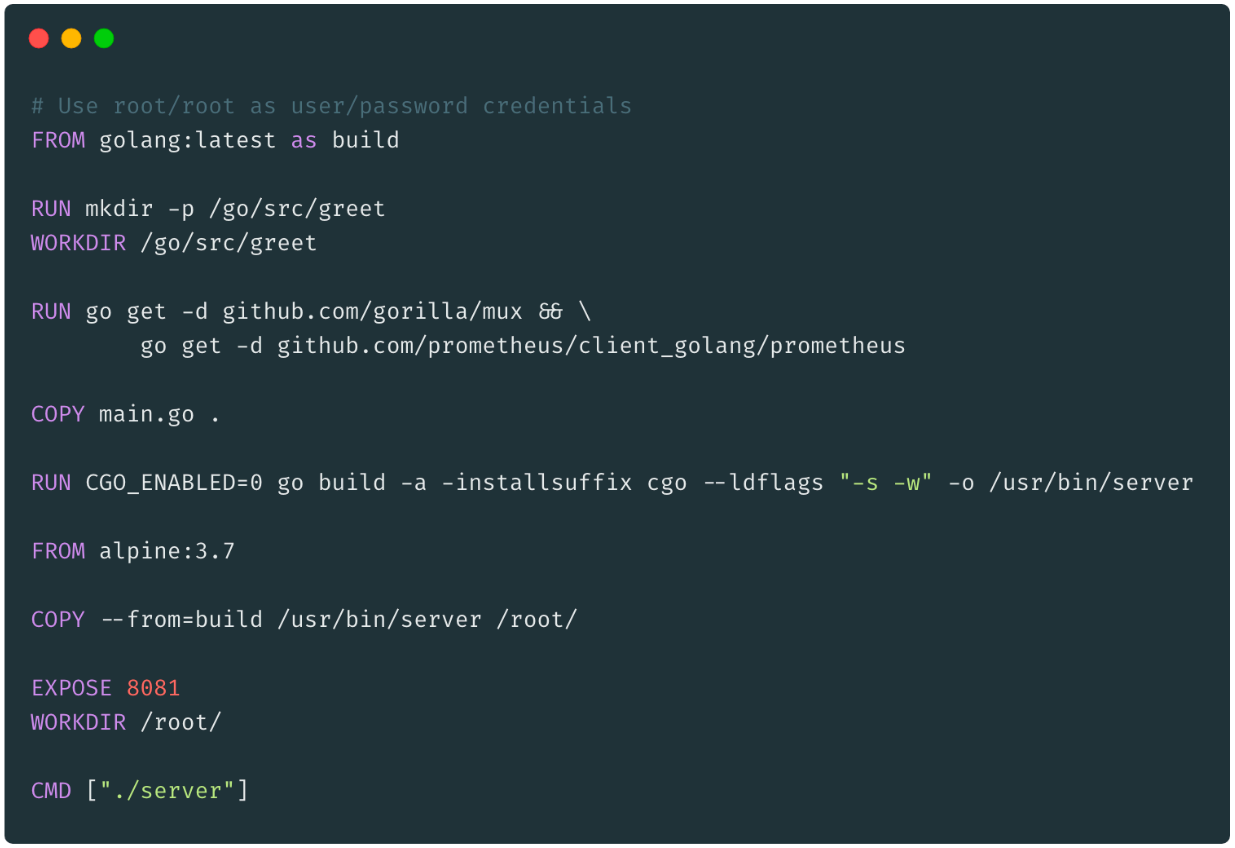
Prometheus Histogram

O **Prometheus Histogram** é ideal para coletar métricas, como: latências de HTTP; número de solicitações; e número total de erros.

Primeiros resultados

```
# HELP go_gc_duration_seconds A summary of the GC invocation durations.
# 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
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 6
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.12"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 547920
# 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 547920
# 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 1.442926e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 64
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 0
# 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 2.240512e+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 547920
# 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 6.504448e+07
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 1.703936e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 1734
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 0
# HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system.
# TYPE go_memstats_heap_sys_bytes gauge
go_memstats_heap_sys_bytes 6.6748416e+07
# HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage collection.
# TYPE go_memstats_last_gc_time_seconds gauge
go_memstats_last_gc_time_seconds 0
# HELP go_memstats_lookups_total Total number of pointer lookups.
# TYPE go_memstats_lookups_total counter
go_memstats_lookups_total 0
# HELP go_memstats_mallocs_total Total number of mallocs.
# TYPE go_memstats_mallocs_total counter
go_memstats_mallocs_total 1798
# HELP go_memstats_mcache_inuse_bytes Number of bytes in use by mcache structures.
# TYPE go_memstats_mcache_inuse_bytes gauge
```

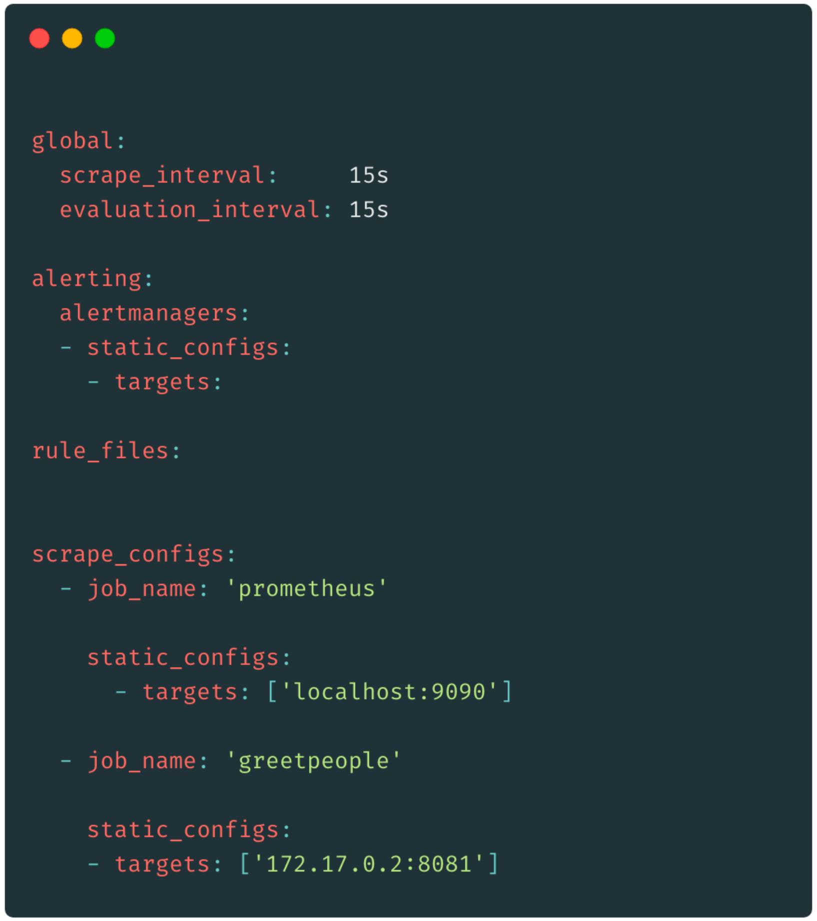
Dockerfile



```dockerfile
# Use root/root as user/password credentials
FROM golang:latest as build

RUN mkdir -p /go/src/greet
WORKDIR /go/src/greet

RUN go get -d github.com/gorilla/mux && \
        go get -d github.com/prometheus/client_golang/prometheus

COPY main.go .

RUN CGO_ENABLED=0 go build -a -installsuffix cgo --ldflags "-s -w" -o /usr/bin/server

FROM alpine:3.7

COPY --from=build /usr/bin/server /root/

EXPOSE 8081
WORKDIR /root/

CMD ["./server"]
```

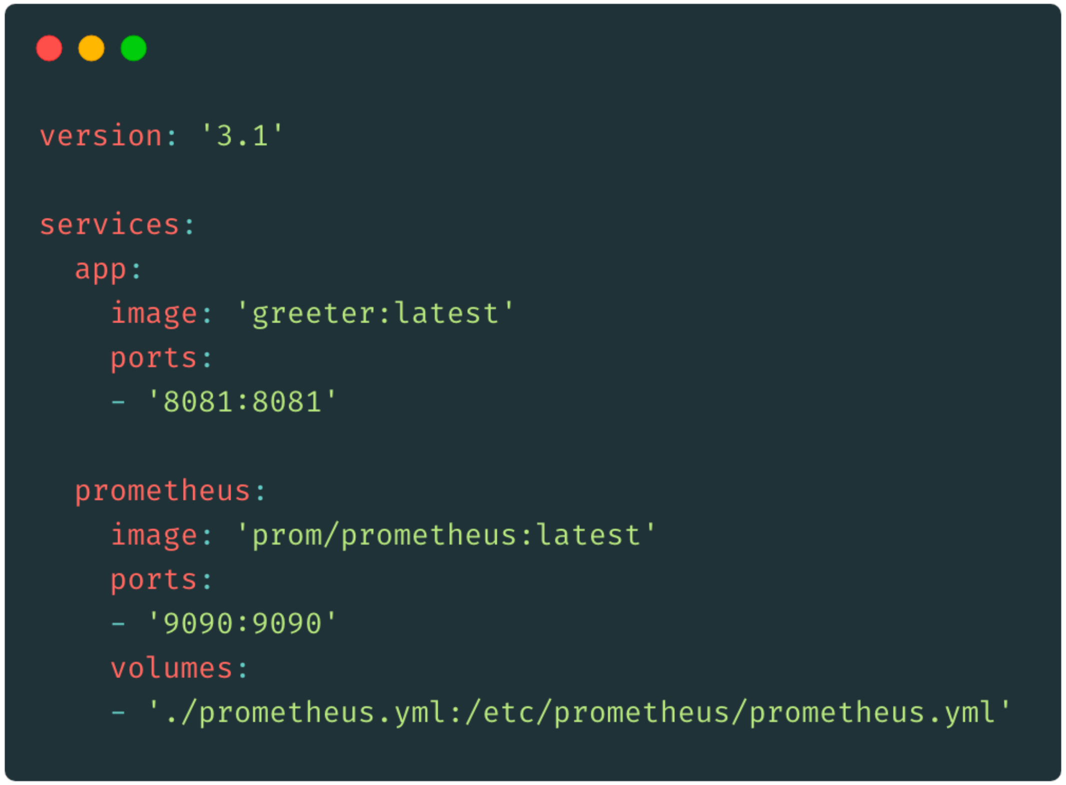
docker build -t greeter .

prometheus.yml



```yaml
global:
  scrape_interval:     15s
  evaluation_interval: 15s

alerting:
  alertmanagers:
  - static_configs:
    - targets:

rule_files:


scrape_configs:
  - job_name: 'prometheus'

    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'greetpeople'

    static_configs:
    - targets: ['172.17.0.2:8081']
```

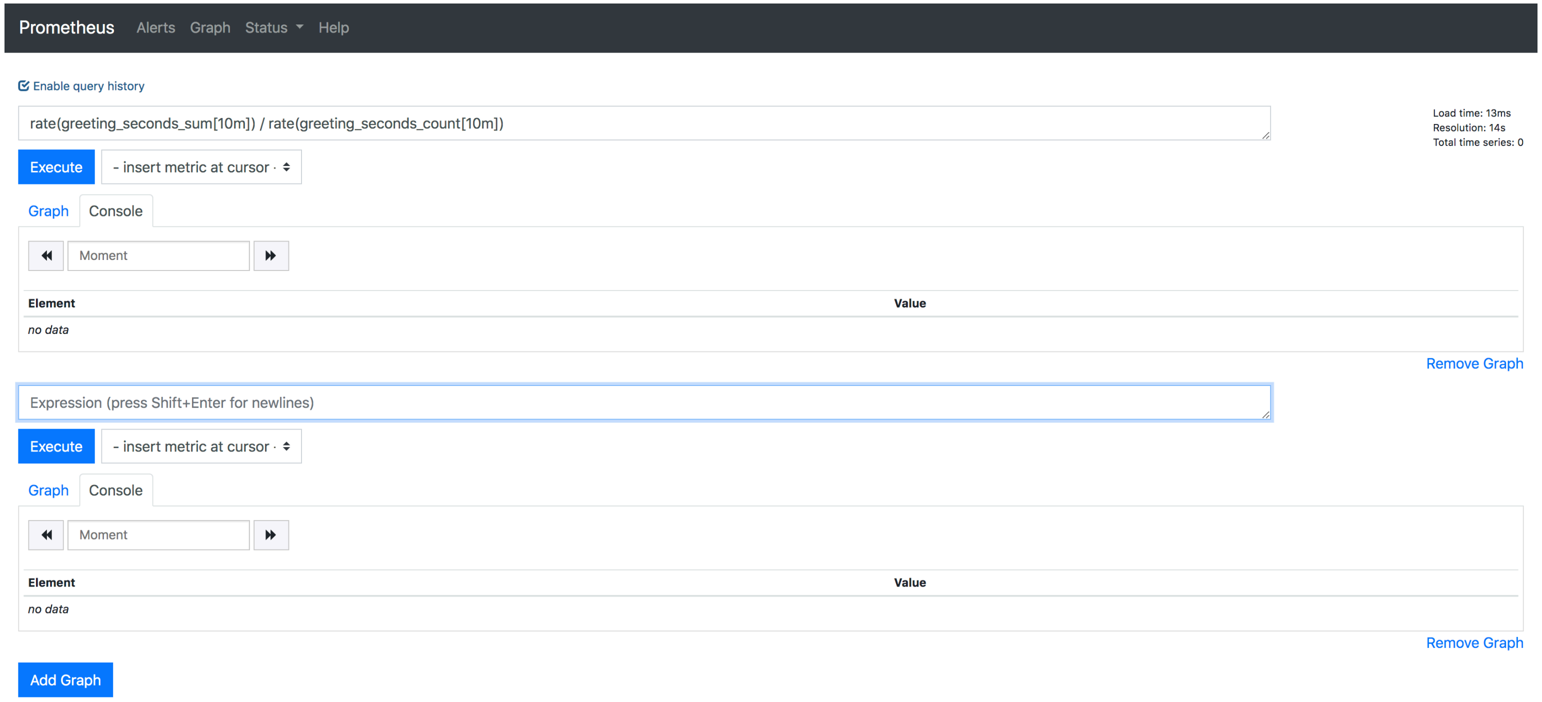
docker-compose.yml



```yaml
version: '3.1'

services:
  app:
    image: 'greeter:latest'
    ports:
    - '8081:8081'

  prometheus:
    image: 'prom/prometheus:latest'
    ports:
    - '9090:9090'
    volumes:
    - './prometheus.yml:/etc/prometheus/prometheus.yml'
```

docker-compose up -d

Medindo



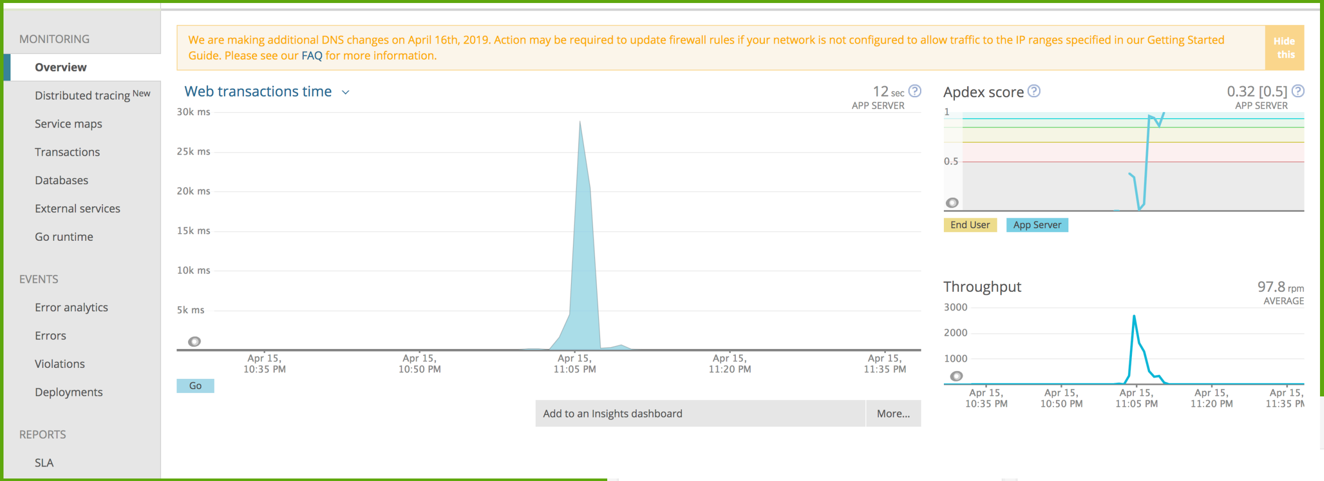
Load time: 13ms
Resolution: 14s
Total time series: 0

```
rate(greeting_seconds_sum[10m]) / rate(greeting_seconds_count[10m])
```

Element	Value
no data	

Element	Value
no data	

New Relic



MONITORING
- Overview
- Distributed tracing New
- Service maps
- Transactions
- Databases
- External services
- Go runtime

EVENTS
- Error analytics
- Errors
- Violations
- Deployments

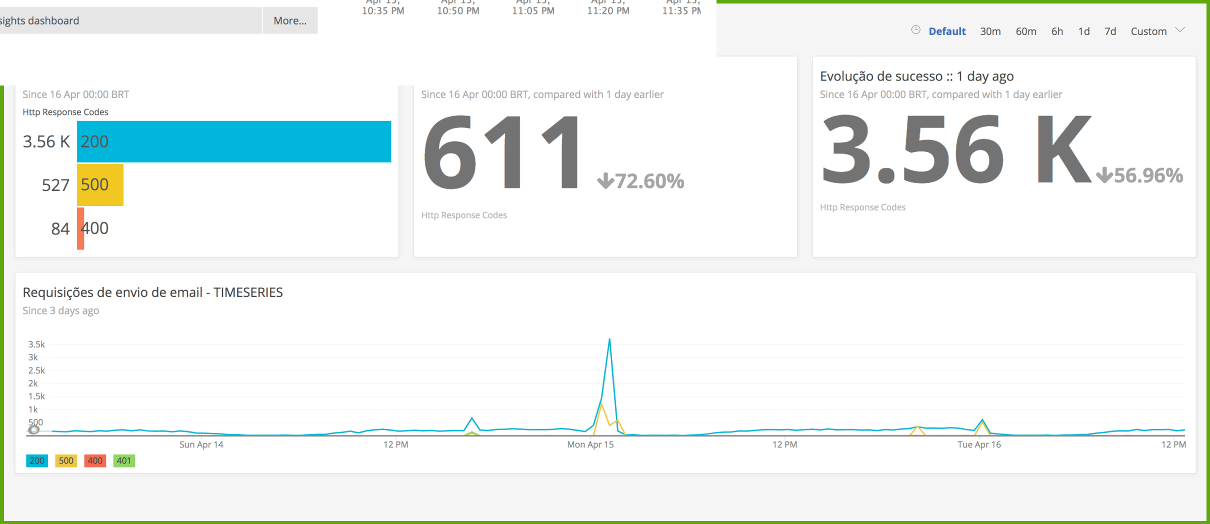
REPORTS
- SLA

We are making additional DNS changes on April 16th, 2019. Action may be required to update firewall rules if your network is not configured to allow traffic to the IP ranges specified in our Getting Started Guide. Please see our FAQ for more information.

Web transactions time — 12 sec APP SERVER

Apdex score — 0.32 [0.5] APP SERVER

Throughput — 97.8 rpm AVERAGE



Since 16 Apr 00:00 BRT

Http Response Codes

3.56 K	200
527	500
84	400

Since 16 Apr 00:00 BRT, compared with 1 day earlier

611 ↓72.60%

Http Response Codes

Evolução de sucesso :: 1 day ago

Since 16 Apr 00:00 BRT, compared with 1 day earlier

3.56 K ↓56.96%

Http Response Codes

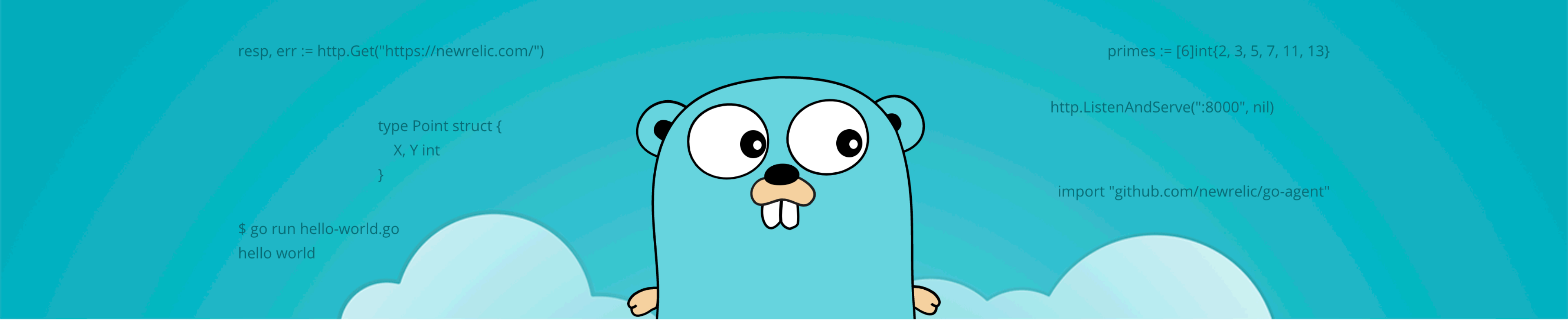
Requisições de envio de email - TIMESERIES

Since 3 days ago

Get the full picture with Go monitoring.

Troubleshoot and solve application performance issues in your complex, highly concurrent Go services.

Sign Up for Go Monitoring



```
resp, err := http.Get("https://newrelic.com/")
```

```
type Point struct {
    X, Y int
}
```

```
$ go run hello-world.go
hello world
```

```
primes := [6]int{2, 3, 5, 7, 11, 13}
```

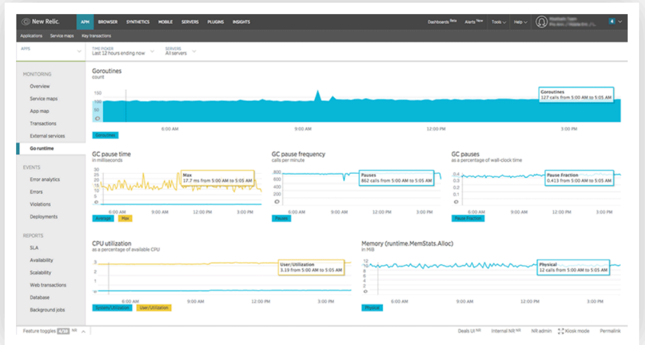
```
http.ListenAndServe(":8000", nil)
```

```
import "github.com/newrelic/go-agent"
```

Go monitoring. Go performance. Go big.

New Relic's Go agent gives you production-level visibility for your Go services.

- See the datastore calls and external services your application is accessing
- Isolate operations that may be causing bottlenecks in responses
- Use deployment markers to see changes in app performance and runtime behavior between deploys
- Write custom events and build custom dashboards with New Relic Insights

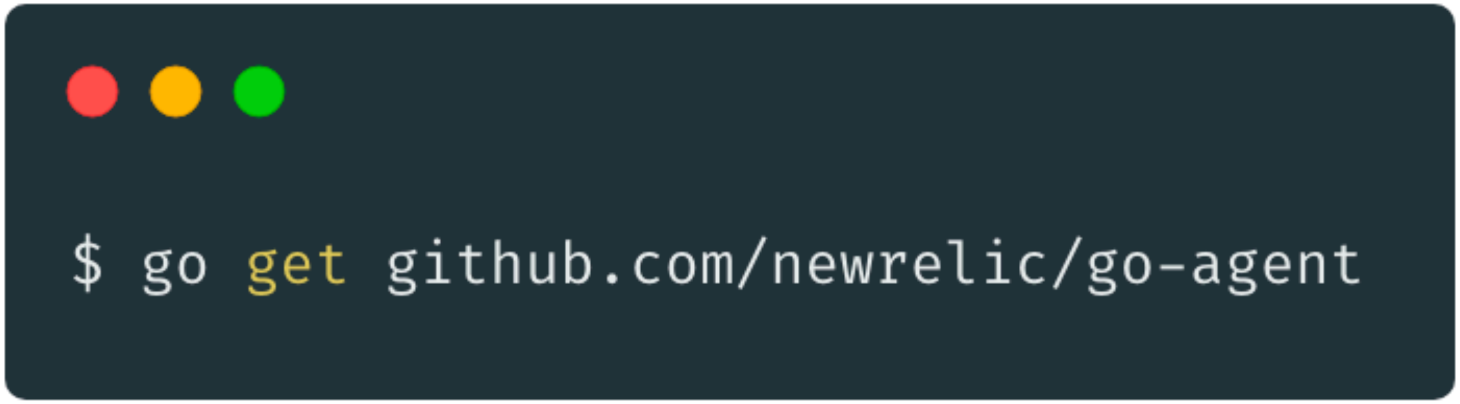


Easily monitor the health of your entire Go runtime environment.

See how your Go applications are performing with full context.

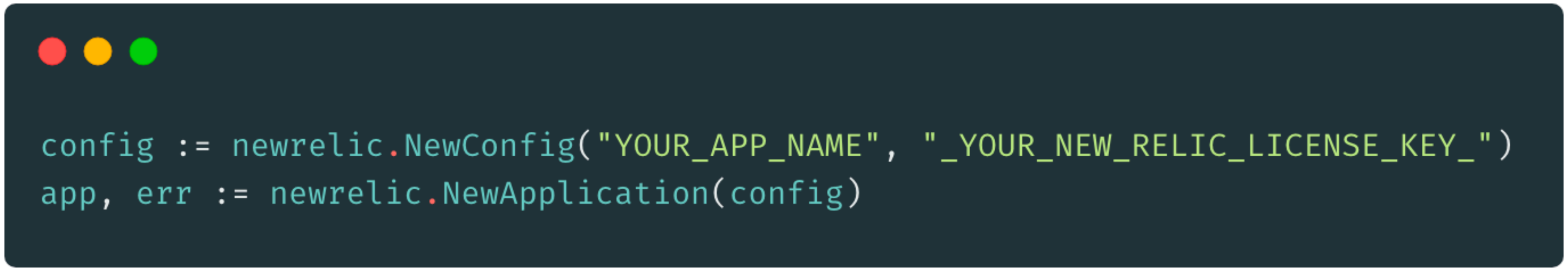
- Track goroutine counts over time in the Go runtime dashboard
- Identify possible Goroutine leaks and narrow down concurrency issues
- Troubleshoot issues using system metrics such as garbage collector information and memory usage

Adicione a dependência



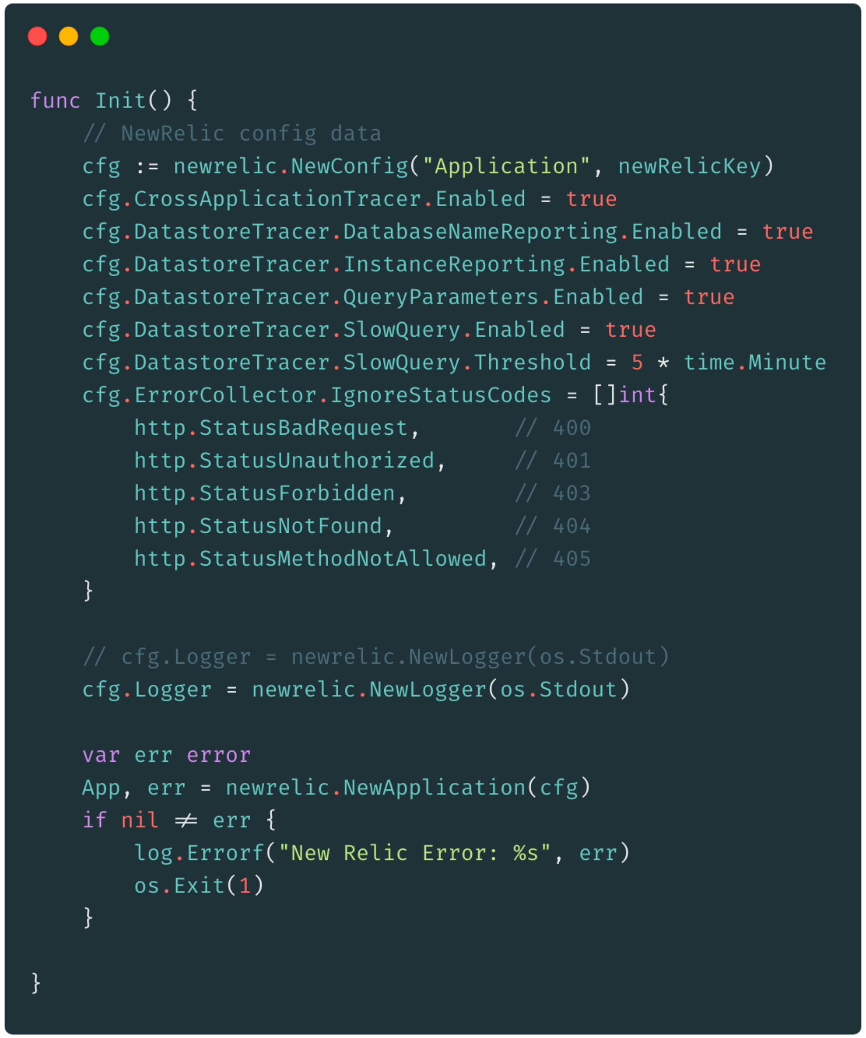
```
$ go get github.com/newrelic/go-agent
```

Prepare seu projeto



```go
config := newrelic.NewConfig("YOUR_APP_NAME", "_YOUR_NEW_RELIC_LICENSE_KEY_")
app, err := newrelic.NewApplication(config)
```

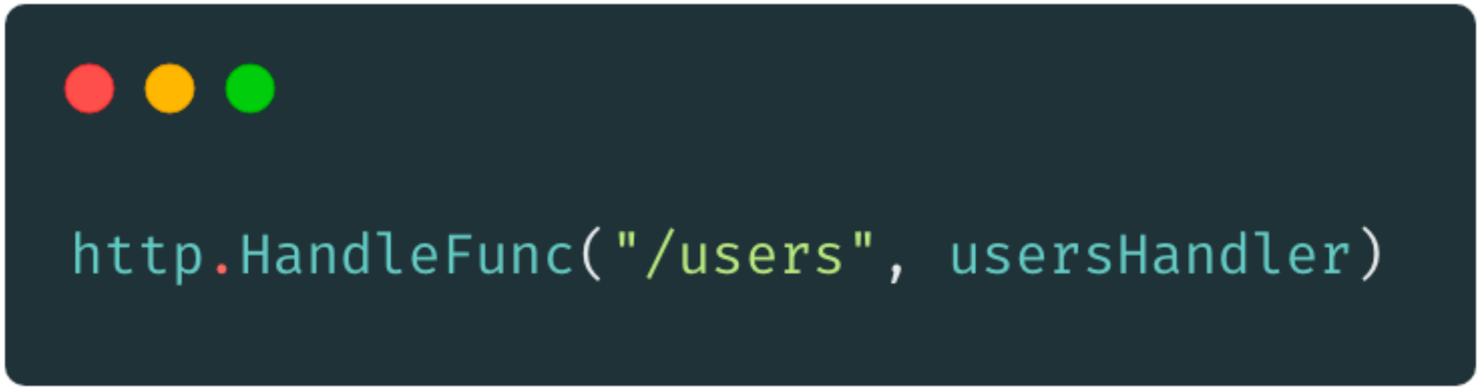
Prepare seu projeto



```go
func Init() {
    // NewRelic config data
    cfg := newrelic.NewConfig("Application", newRelicKey)
    cfg.CrossApplicationTracer.Enabled = true
    cfg.DatastoreTracer.DatabaseNameReporting.Enabled = true
    cfg.DatastoreTracer.InstanceReporting.Enabled = true
    cfg.DatastoreTracer.QueryParameters.Enabled = true
    cfg.DatastoreTracer.SlowQuery.Enabled = true
    cfg.DatastoreTracer.SlowQuery.Threshold = 5 * time.Minute
    cfg.ErrorCollector.IgnoreStatusCodes = []int{
        http.StatusBadRequest,       // 400
        http.StatusUnauthorized,     // 401
        http.StatusForbidden,        // 403
        http.StatusNotFound,         // 404
        http.StatusMethodNotAllowed, // 405
    }

    // cfg.Logger = newrelic.NewLogger(os.Stdout)
    cfg.Logger = newrelic.NewLogger(os.Stdout)

    var err error
    App, err = newrelic.NewApplication(cfg)
    if nil ≠ err {
        log.Errorf("New Relic Error: %s", err)
        os.Exit(1)
    }

}
```

github.com/newrelic/go-agent

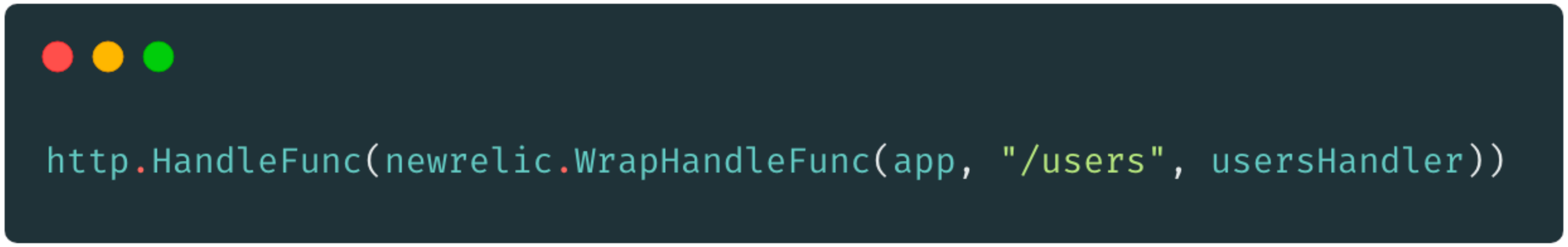
Metrifique suas requisições



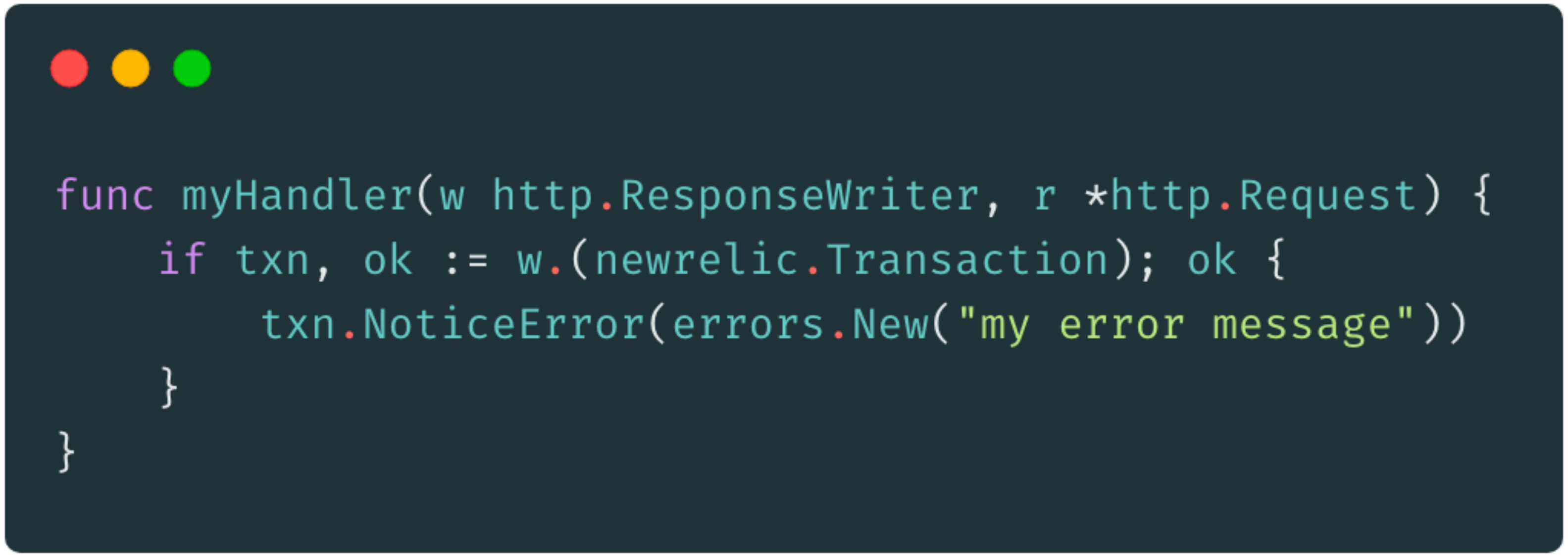
```go
http.HandleFunc("/users", usersHandler)
```

HTTP Handler Padrão

Metrifique suas requisições

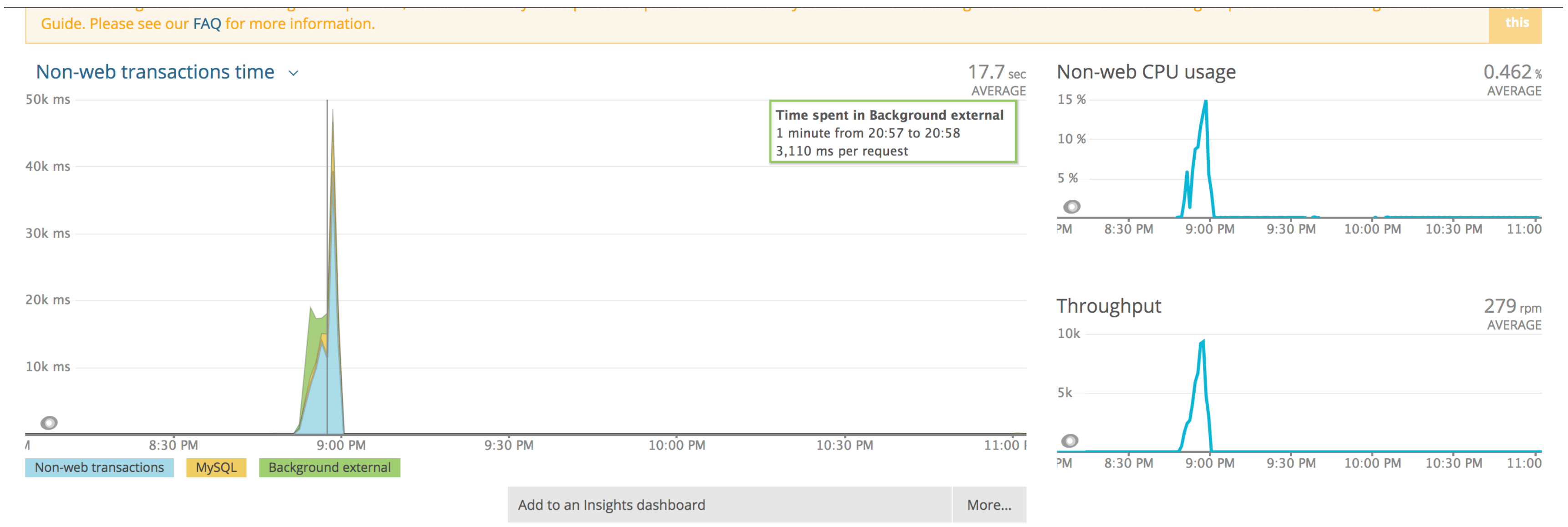


```go
http.HandleFunc(newrelic.WrapHandleFunc(app, "/users", usersHandler))
```

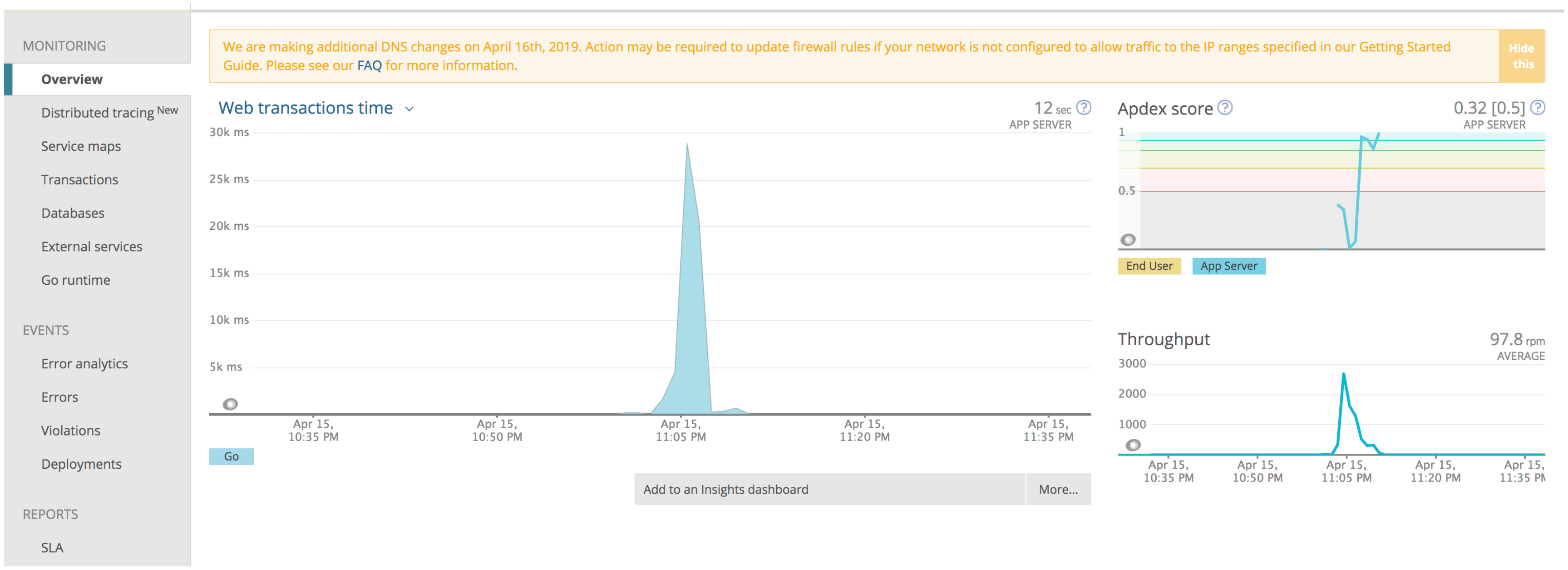


```go
func myHandler(w http.ResponseWriter, r *http.Request) {
    if txn, ok := w.(newrelic.Transaction); ok {
        txn.NoticeError(errors.New("my error message"))
    }
}
```

APM


Guide. Please see our FAQ for more information.
this
Non-web transactions time
17.7 sec AVERAGE
50k ms
40k ms
30k ms
20k ms
10k ms
8:30 PM
9:00 PM
9:30 PM
10:00 PM
10:30 PM
11:00
Time spent in Background external
1 minute from 20:57 to 20:58
3,110 ms per request
Non-web transactions
MySQL
Background external
Add to an Insights dashboard
More...
Non-web CPU usage
0.462 % AVERAGE
15 %
10 %
5 %
Throughput
279 rpm AVERAGE
10k
5k

APM



MONITORING

Overview

Distributed tracing New

Service maps

Transactions

Databases

External services

Go runtime

EVENTS

Error analytics

Errors

Violations

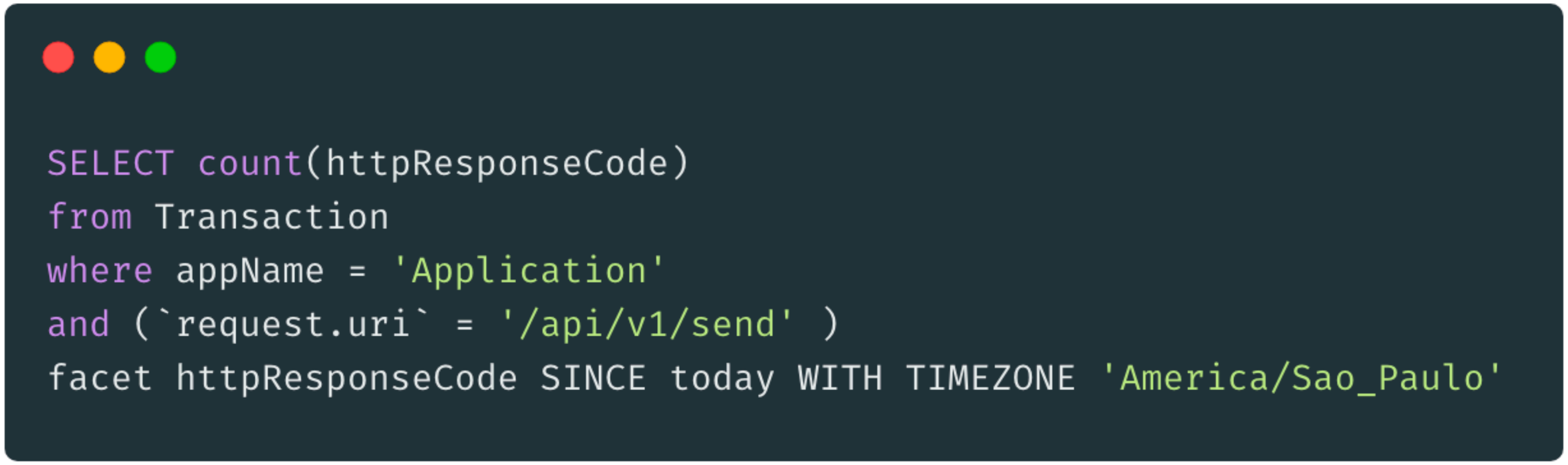
Deployments

REPORTS

SLA

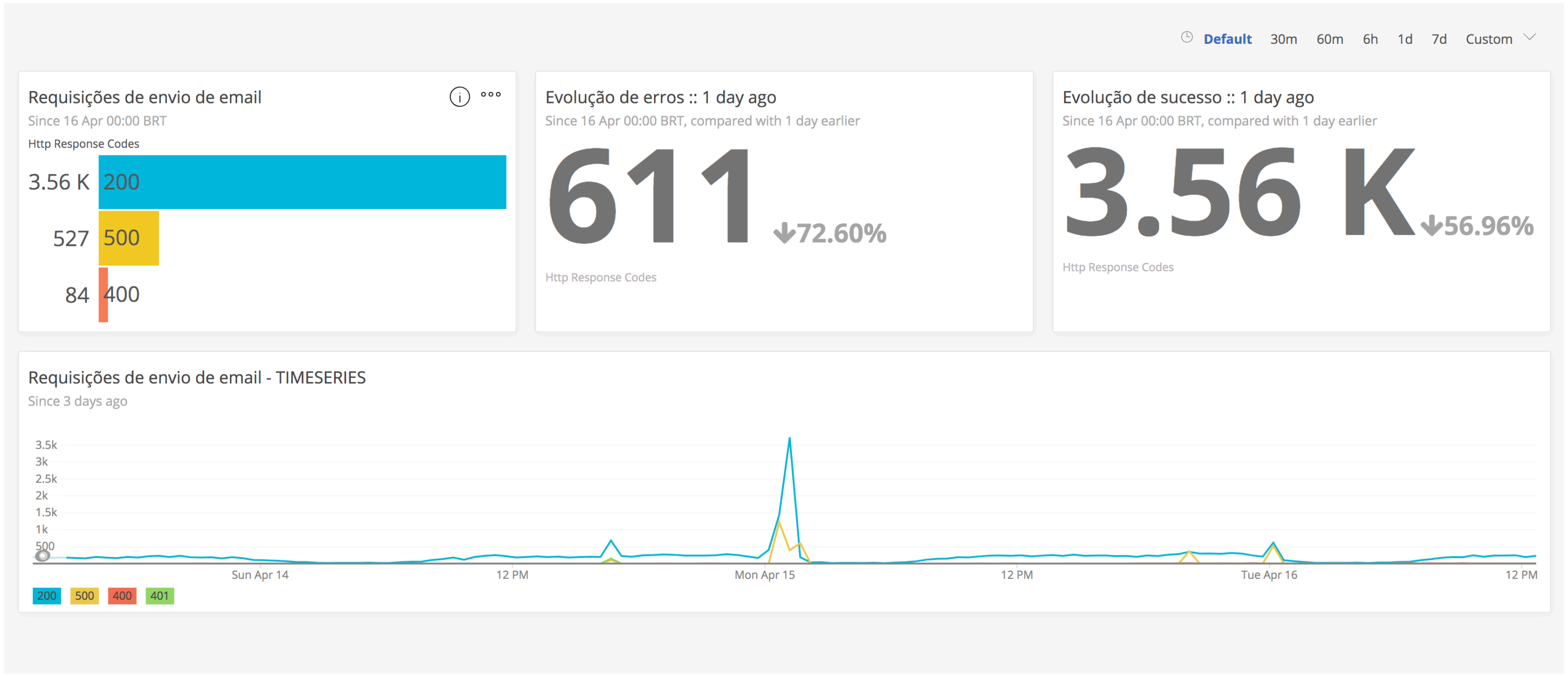
We are making additional DNS changes on April 16th, 2019. Action may be required to update firewall rules if your network is not configured to allow traffic to the IP ranges specified in our Getting Started Guide. Please see our FAQ for more information.

Insights



```
SELECT count(httpResponseCode)
from Transaction
where appName = 'Application'
and (`request.uri` = '/api/v1/send' )
facet httpResponseCode SINCE today WITH TIMEZONE 'America/Sao_Paulo'
```

Insights



Requisições de envio de email

Since 16 Apr 00:00 BRT

Http Response Codes

3.56 K	200
527	500
84	400

Evolução de erros :: 1 day ago

Since 16 Apr 00:00 BRT, compared with 1 day earlier

611 ↓72.60%

Http Response Codes

Evolução de sucesso :: 1 day ago

Since 16 Apr 00:00 BRT, compared with 1 day earlier

3.56 K ↓56.96%

Http Response Codes

Requisições de envio de email - TIMESERIES

Since 3 days ago

Regra de ouro #3

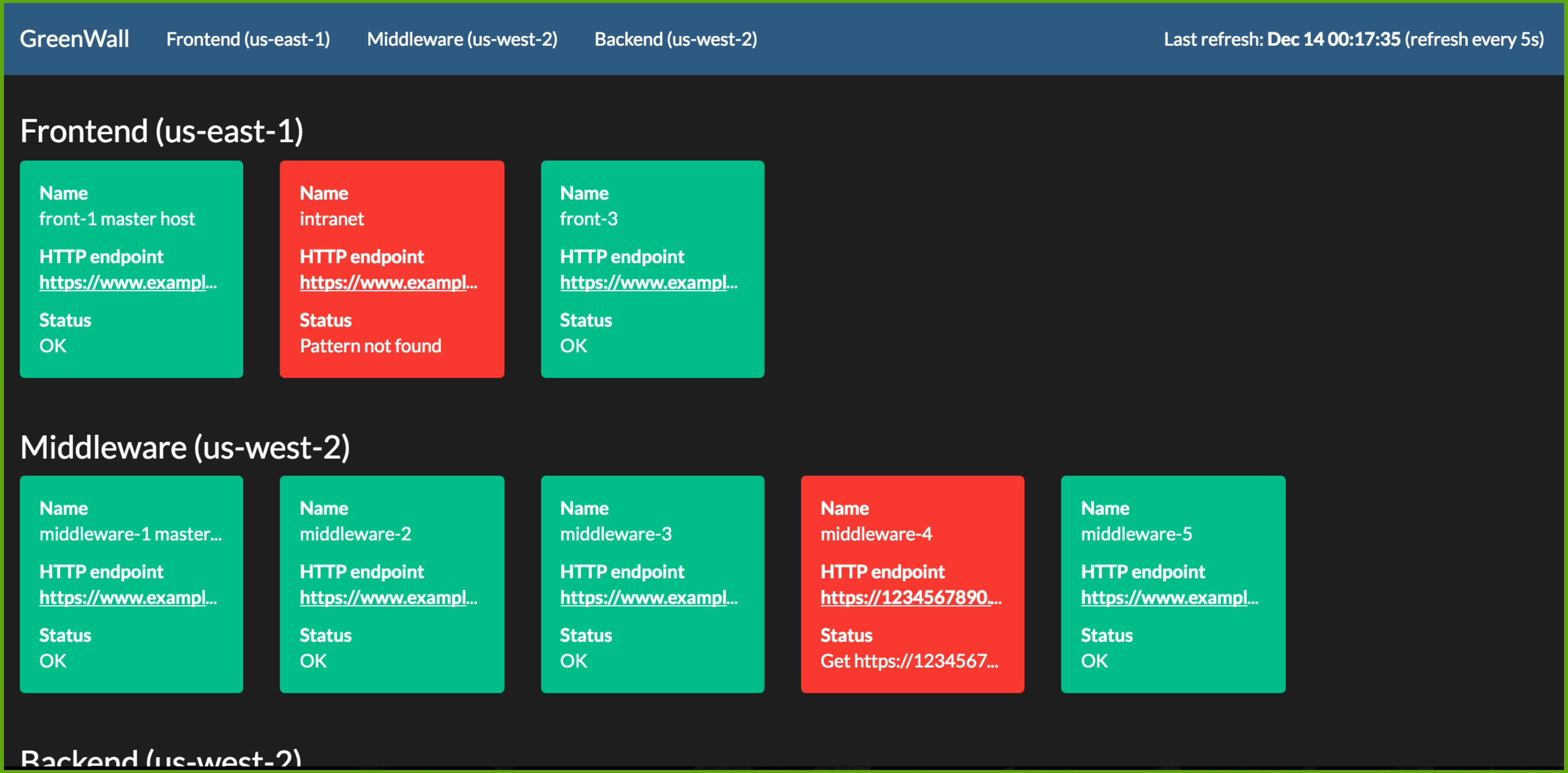
Crie painéis de monitoramento que façam sentido

Homer's Web Page

Diagnostique rápido, medique rápido

Esteja sempre atendo, monitore e investigue quando algo parecer estranho

GreenWall



GreenWall Frontend (us-east-1) Middleware (us-west-2) Backend (us-west-2) Last refresh: **Dec 14 00:17:35** (refresh every 5s)

Frontend (us-east-1)

Name
front-1 master host
HTTP endpoint
https://www.exampl...
Status
OK

Name
intranet
HTTP endpoint
https://www.exampl...
Status
Pattern not found

Name
front-3
HTTP endpoint
https://www.exampl...
Status
OK

Middleware (us-west-2)

Name
middleware-1 master...
HTTP endpoint
https://www.exampl...
Status
OK

Name
middleware-2
HTTP endpoint
https://www.exampl...
Status
OK

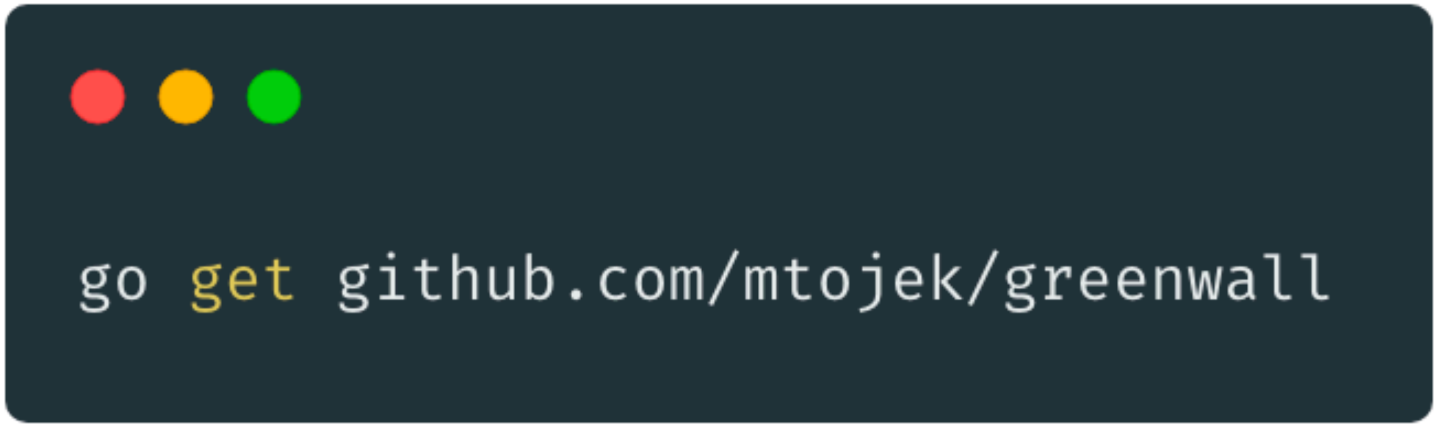
Name
middleware-3
HTTP endpoint
https://www.exampl...
Status
OK

Name
middleware-4
HTTP endpoint
https://1234567890...
Status
Get https://1234567...

Name
middleware-5
HTTP endpoint
https://www.exampl...
Status
OK

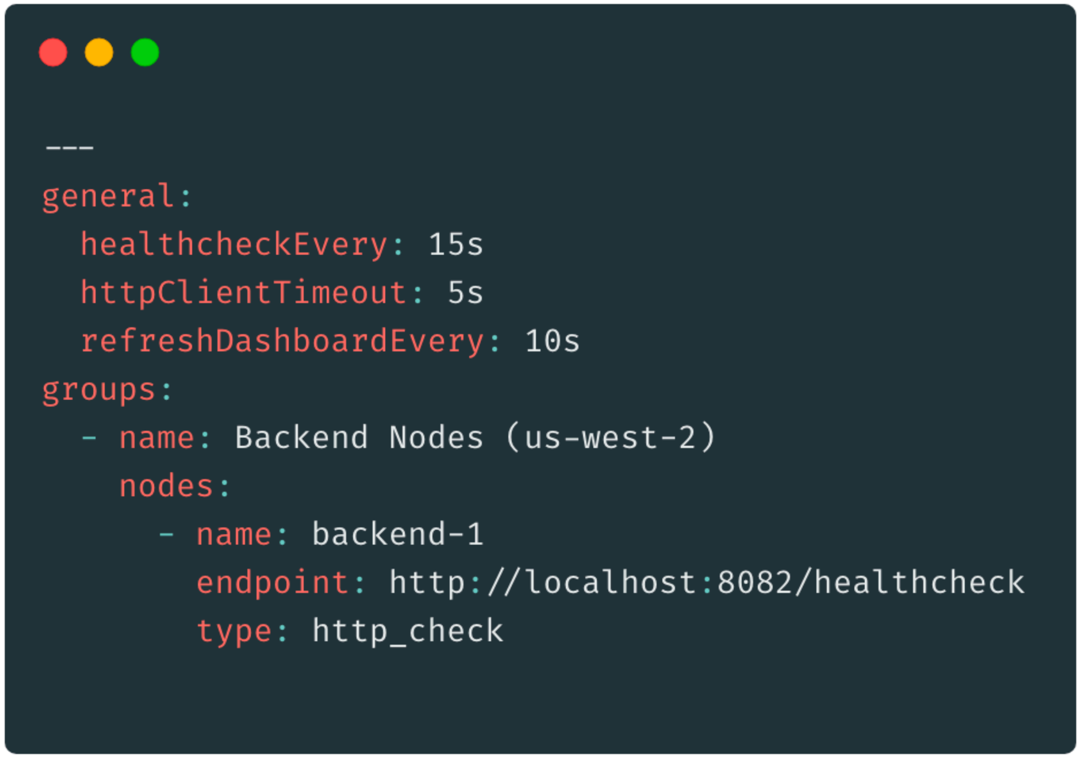
Backend (us-west-2)

Instalando



```
go get github.com/mtojek/greenwall
```

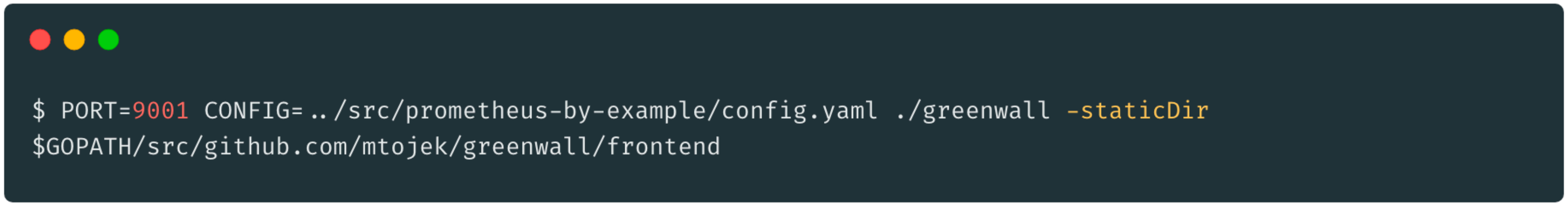
Configurando



```yaml
---
general:
  healthcheckEvery: 15s
  httpClientTimeout: 5s
  refreshDashboardEvery: 10s
groups:
  - name: Backend Nodes (us-west-2)
    nodes:
      - name: backend-1
        endpoint: http://localhost:8082/healthcheck
        type: http_check
```

github.com/mtojek/greenwall

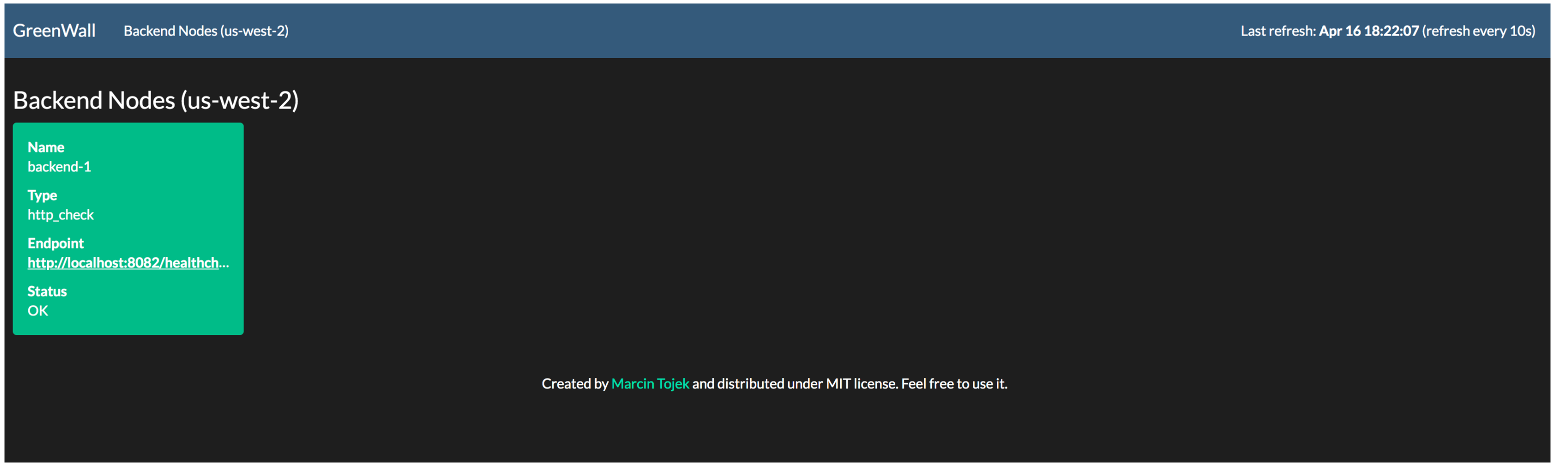
Executando



```
$ PORT=9001 CONFIG=../src/prometheus-by-example/config.yaml ./greenwall -staticDir
$GOPATH/src/github.com/mtojek/greenwall/frontend
```

github.com/mtojek/greenwall

Monitorando


GreenWall Backend Nodes (us-west-2) Last refresh: **Apr 16 18:22:07** (refresh every 10s)

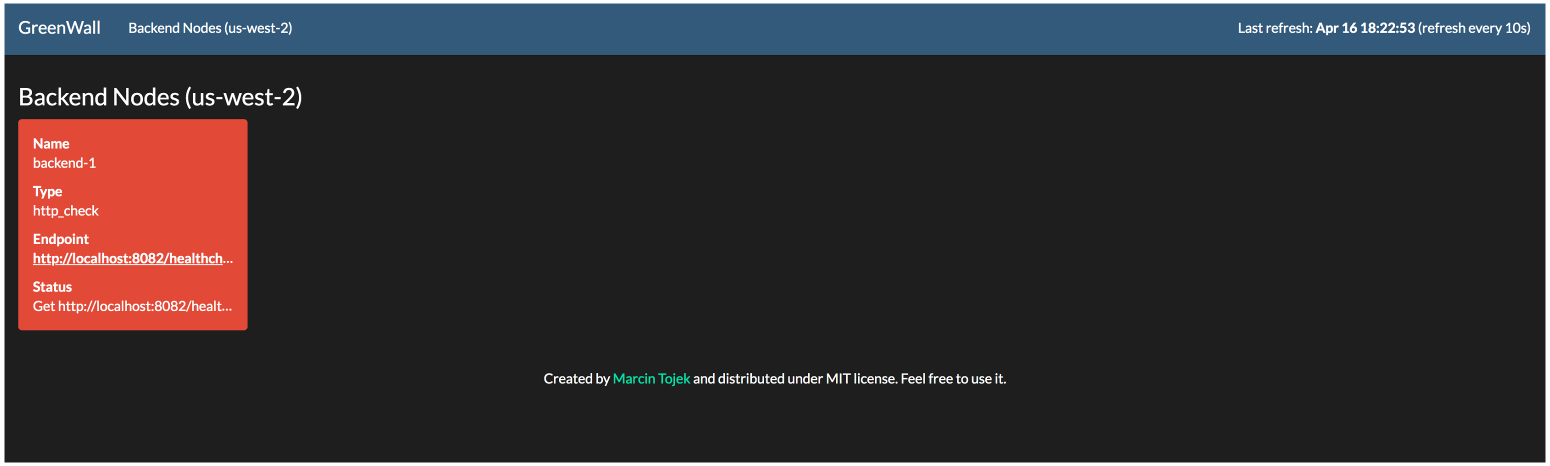
Backend Nodes (us-west-2)

Name
backend-1
Type
http_check
Endpoint
http://localhost:8082/healthch...
Status
OK

Created by Marcin Tojek and distributed under MIT license. Feel free to use it.

github.com/mtojek/greenwall

Monitorando



GreenWall Backend Nodes (us-west-2) Last refresh: **Apr 16 18:22:53** (refresh every 10s)

Backend Nodes (us-west-2)

Name
backend-1

Type
http_check

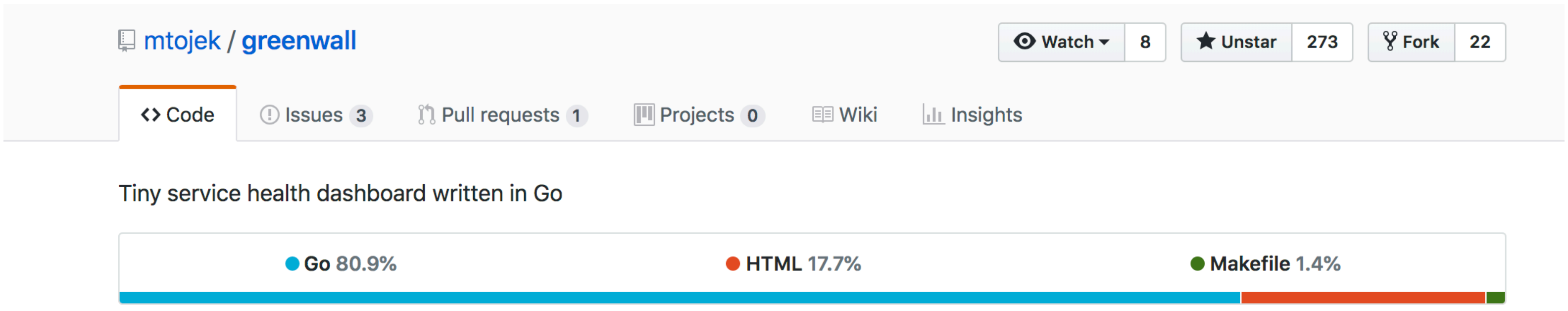
Endpoint
http://localhost:8082/healthch...

Status
Get http://localhost:8082/healt...

Created by Marcin Tojek and distributed under MIT license. Feel free to use it.

github.com/mtojek/greenwall

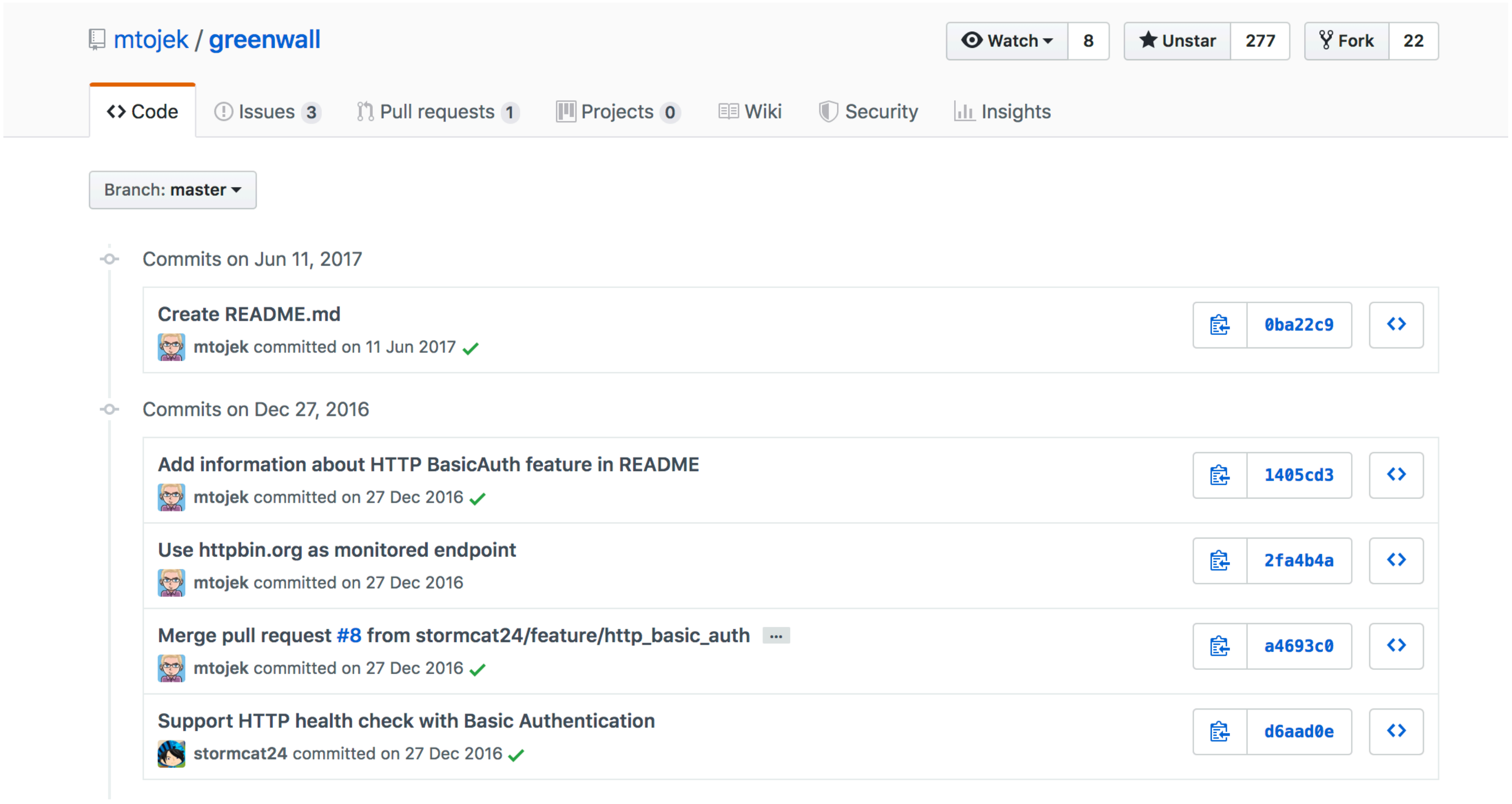
Escrito em Go


mtojek / greenwall

Tiny service health dashboard written in Go

github.com/mtojek/greenwall

Só tem um problema...



mtojek / greenwall

Watch 8 | Unstar 277 | Fork 22

Code | Issues 3 | Pull requests 1 | Projects 0 | Wiki | Security | Insights

Branch: master

Commits on Jun 11, 2017

Create README.md
mtojek committed on 11 Jun 2017 ✓ — 0ba22c9

Commits on Dec 27, 2016

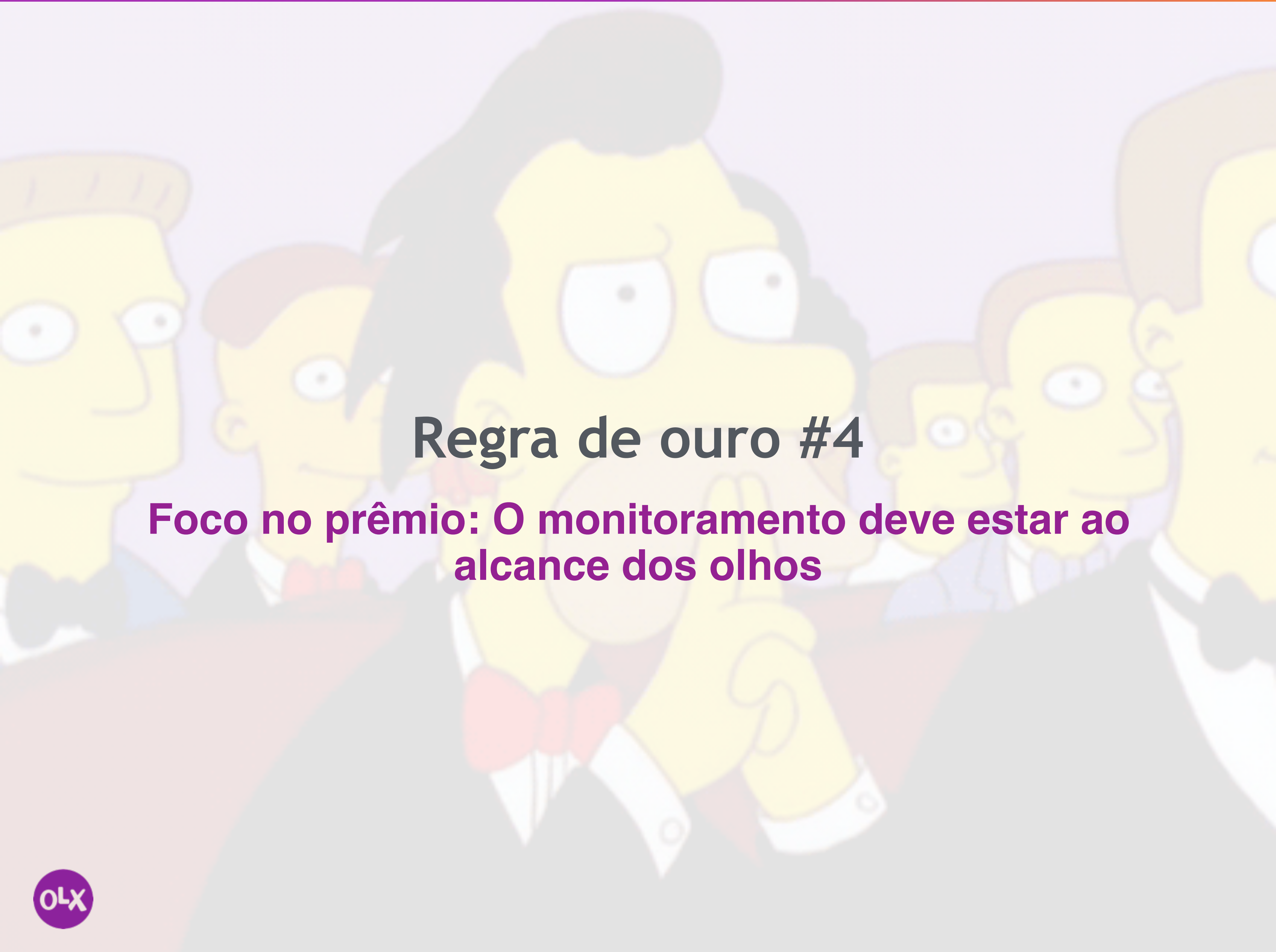
Add information about HTTP BasicAuth feature in README
mtojek committed on 27 Dec 2016 ✓ — 1405cd3

Use httpbin.org as monitored endpoint
mtojek committed on 27 Dec 2016 — 2fa4b4a

Merge pull request #8 from stormcat24/feature/http_basic_auth …
mtojek committed on 27 Dec 2016 ✓ — a4693c0

Support HTTP health check with Basic Authentication
stormcat24 committed on 27 Dec 2016 ✓ — d6aad0e

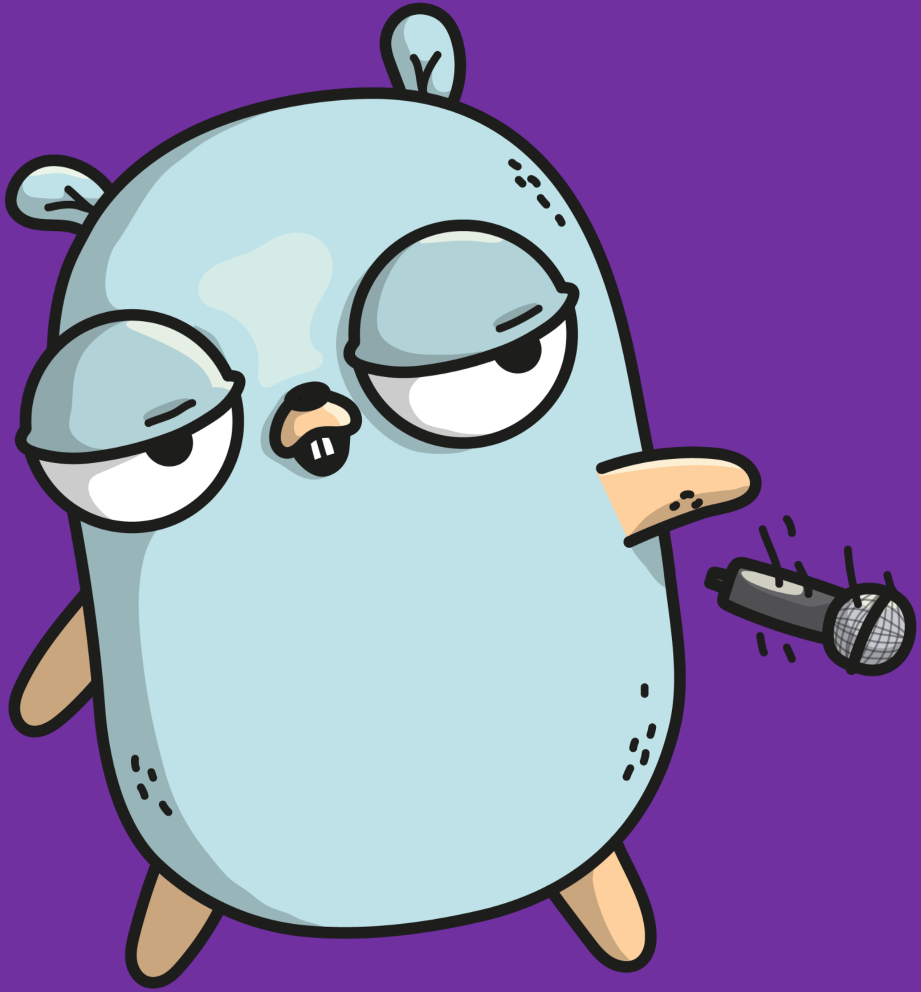
github.com/mtojek/greenwall



Regra de ouro #4

Foco no prêmio: O monitoramento deve estar ao alcance dos olhos

That’s It =)



QUESTIONS.... ANYBODY?
memegenerator.net

Sigam-me =)

@marcopollivier

ollivier.com.br

OH, THAT'S GREAT.
THANK YOU. THANKS A LOT.