



...

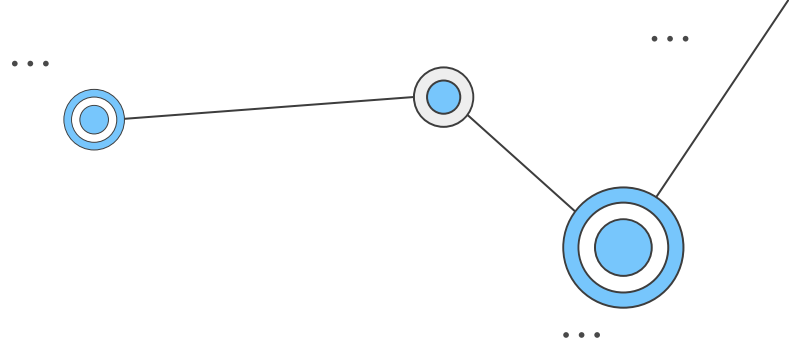


0 que aprendemos
usando Go durante 1 ano em
ambiente serverless?

Gabriel Prando



Gabriel Prando



Engenheiro de software na **Conta Simples**.

Backend no time de plataforma, engenheiro de computação formado pela UTFPR-PR, futuro mestre em Engenharia de Computação.

Conteúdo

01

Visão sobre Golang e Nodejs

02

Vantagens e desvantagens

03

Porque começamos a utilizar

04

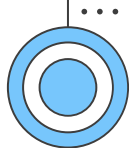
Resultados e comparações



01

Golang e Nodejs

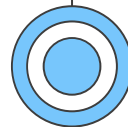
Breve visão



...



...



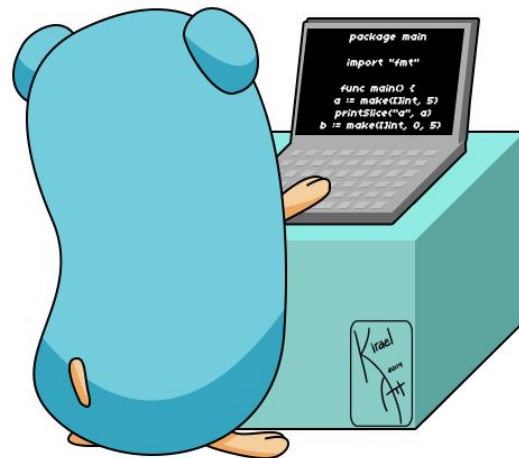
...

Breve visão

	Golang	Nodejs
Criação	2007 (2009 código aberto)	2009
Versão LTS atual	1.21.0	18.17.1
Linguagens base	Go, Assembly e C++	C++, C e Javascript
Plataforma	Multiplataforma	Multiplataforma
Disponibilidade de devs	Média/Baixa	Alta
Tipo	Compilada	Interpretada
Comunidade	Crescente	Grande
Paradigma	Estruturada/Orientada	Multiparadigma
Palavras reservadas	25	64

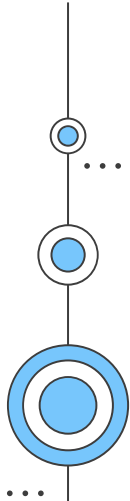
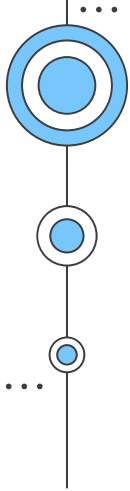
Casos de uso Golang

- Programas Concorrentes e Paralelos;
- Sistemas de Rede e Servidores;
- Microsserviços e CLIs;
- Aplicativos de Tempo Real;
- Automações, backend...



02

Vantagens e desvantagens

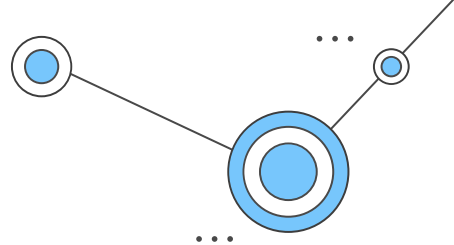


Vantagens e desvantagens

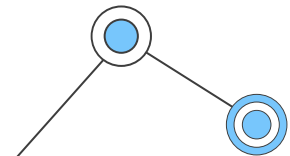
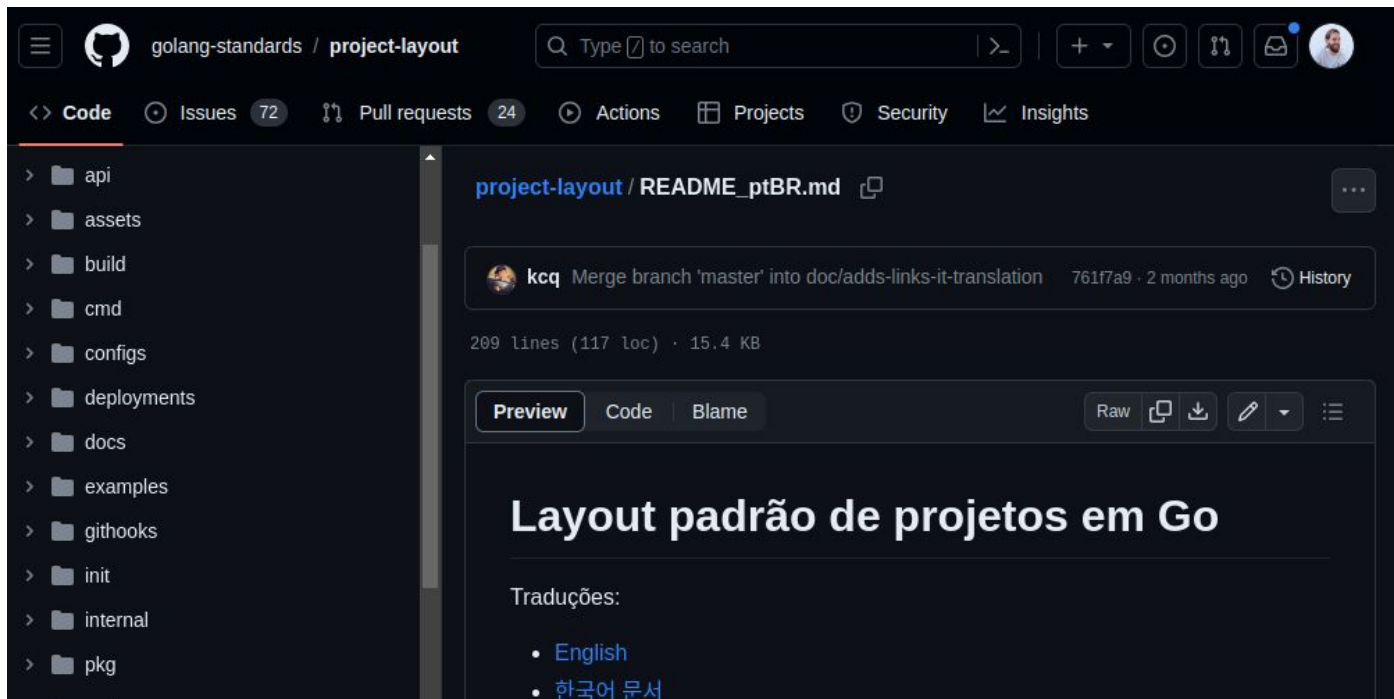
- + Fácil de entender;
- + Segura;
- + Simplicidade;
- + Alta performance;
- + Fácil execução.

- Comunidade ainda pequena;
- Poucos pacotes terceiros;
- Arquiteturas e padrões poucos definidos.

Estrutura em projetos

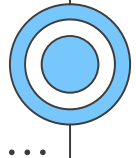
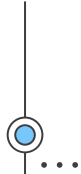
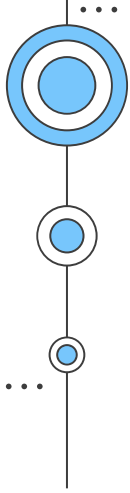


Mais de 41k de stars.



03

Por que começamos
a utilizar?





Motivações

01

Necessidade de baixa
latência e cold start

02

Aposta na linguagem

03

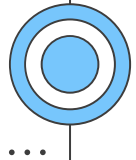
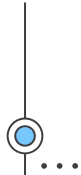
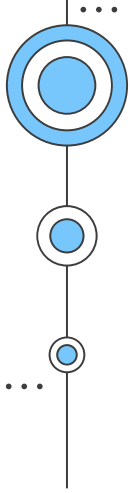
Confiabilidade

04

Interesse do time

04

Resultados e comparações





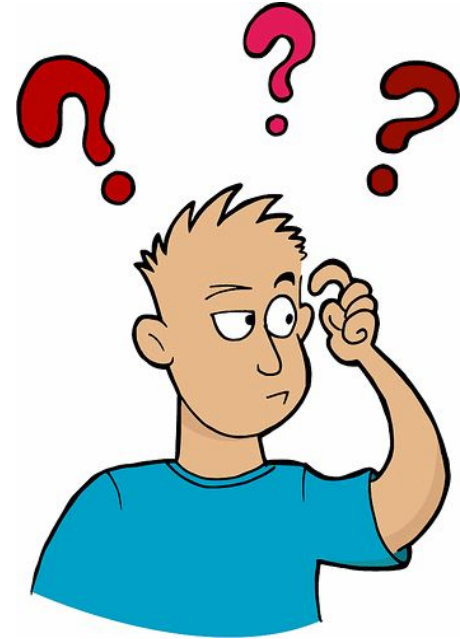
Métricas levantadas

- Dois serviços "similares";
- Um em node e outro em Golang;
- Período de uma semana;
- Mais de 1 milhão de invocações.

...

Métrica Percentiles

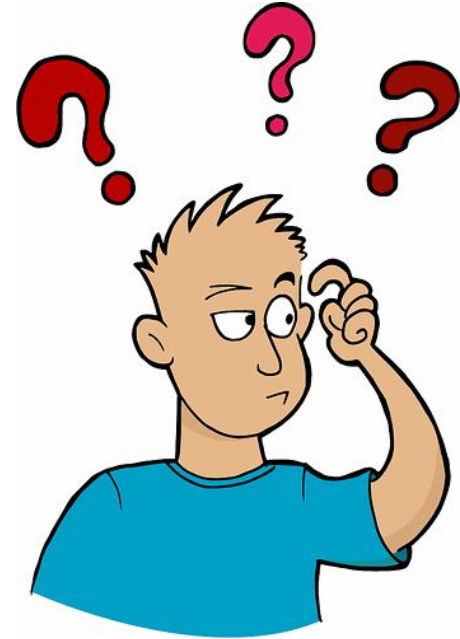
- Ordena uma amostra de forma crescente e divide em 100 partes;
- Cada uma com uma porcentagem dos dados aproximadamente igual.



Exemplo Percentil

[20, 30, 40, 50, 60, 70, 800, 900]

- (P50= 60ms): descartamos 50% (em ordem) e recuperamos o primeiro valor que “sobra”, logo 50% dos casos tiveram até 60ms de tempo;
- P90: descartamos 90%, no caso o P90 = 900ms.



Métrica Percentiles Lambda

```
filter @type="REPORT"  
| fields greatest(@initDuration, 0) + @duration as duration, ispresent(@initDuration) as coldStart  
| stats count(*) as count, pct(duration, 50) as p50, pct(duration, 90) as p90, pct(duration, 99) as p99, max(duration) as max by coldStart
```


Percentiles com e sem cold start

	Count	p50	p90	p99	max
Golang	3.427.538	10,95	90,40	397,32	5007,37
Nodejs	1.383.811	22,44	2934,50	2981,45	4498,14
Golang coldStart	4.759	862,31	953,91	1121,57	3007,59
Nodejs coldstart	2.677	4749,37	4938,16	5058,05	5333,56

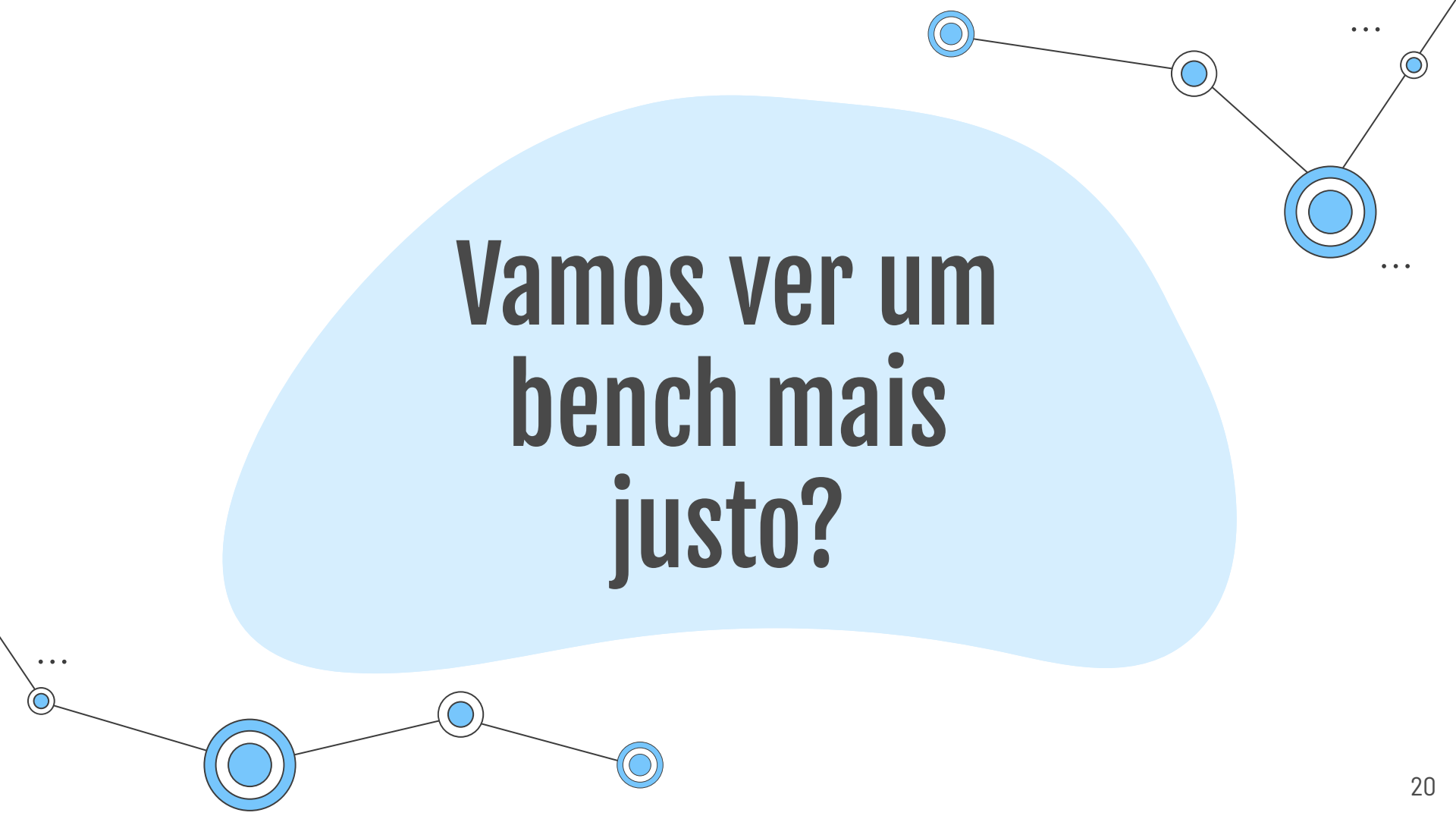
Métrica de memória utilizada

Média, mínima e máxima.

```
filter @type="REPORT"  
| stats avg(@maxMemoryUsed/1024/1024) as mean_MemoryUsed,  
... min(@maxMemoryUsed/1024/1024) as min_MemoryUsed,  
... max(@maxMemoryUsed/1024/1024) as max_MemoryUsed
```

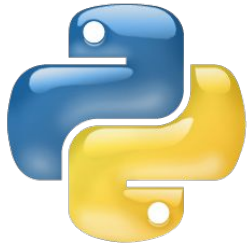
Memória utilizada

	Média (MB)	Mínima (MB)	Máxima (MB)
Golang	60,91	28,61	68,60
Nodejs	144,08	128,74	168,80



**Vamos ver um
bench mais
justo?**

Cenário



vs



vs



- Serverless framework;
- Python 3.9, Go 1.X e Node.js 18.x;
- 1024MB de memória;
- API Gateway V2 + lambda;
- Testes de carga com K6 e grafana cloud para report.

Cenário

GET /users/{id}



endpoint



API Gateway

description



{language}-get-user

Get user by ID



User

users table

Tamanho dos pacotes

	Tamanho (MB)	Runtime
Golang	4,1	Go1.X
Nodejs	2,6	Node 18
Python	56,0	Python 3.9

K6



```
1 stages: [  
2   { duration: "5s", target: "10" },  
3   { duration: "20s", target: "50" },  
4   { duration: "15s", target: "30" },  
5   { duration: "10s", target: "20" },  
6   { duration: "5s", target: "10" },  
7   { duration: "5s", target: "5" },  
8 ],
```


K6

```
1 let res = http.get(process.env.PYTHON_URL);
2
3 GetPythonDuration.add(res.timings.duration);
4 PythonRequisitionsRate.add(1);
5 PythonFailRate.add(res.status === 0 || res.status > 399);
6 PythonSuccessRate.add(res.status < 399);
7
8 if (
9   !check(res, {
10     "is statuscode 200 - endpoint user": (r) => r.status === 200,
11   })
12 ) {
13   fail(durationMsg);
14 }
```

Resultados gerais

🕒 Request Metrics

📊 Other Stats

☰ Checks & Groups

Checks

Passed 1368
Failed 0

Iterations

Total 456
Rate 7.52/s

Virtual Users

Min 2
Max 50

Requests

Total 1368
Rate 22.57/s

Data Received

Total 1.16 MB
Rate 0.02 mB/s

Data Sent

Total 0.15 MB
Rate 0.00 mB/s

Resultados gerais

Total Requests

1368

Failed Requests

0

Breached Thresholds

0

Failed Checks

0

🕒 Request Metrics

📊 Other Stats

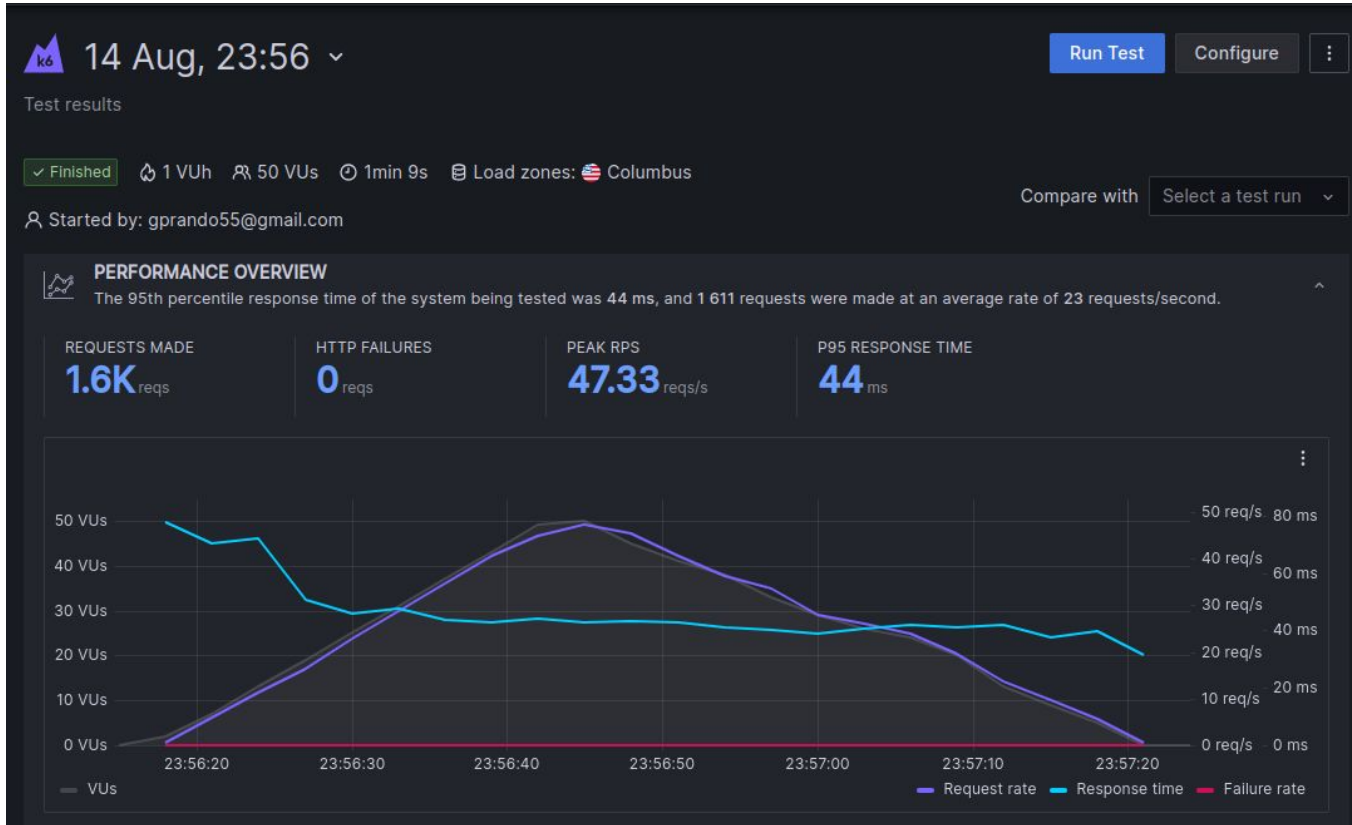
☰ Checks & Groups

	Count	Rate	Average	Maximum	Median	Minimum	90th Percentile	95th Percentile
http_req_duration	-	-	171.27	1367.49	158.13	144.29	173.08	182.60
http_req_waiting	-	-	170.60	1367.26	157.87	144.16	172.72	182.32
http_req_connecting	-	-	15.25	252.00	-	-	131.14	135.08
http_req_tls_handshaking	-	-	30.73	424.31	-	-	268.57	275.10

Resultados por lambda

	Count	Rate	Average	Maximum	Median	Minimum	90th Percentile	95th Percentile
get_golang_duration	-	-	164.56	880.63	156.49	145.51	171.24	181.09
get_nodejs_duration	-	-	170.21	1052.27	157.20	144.29	172.81	183.55
get_python_duration	-	-	179.04	1367.49	160.03	148.44	175.55	185.82

K6 cloud



Percentiles com e sem cold start

	Count	P50 (ms)	p90(ms)	p99(ms)	max(ms)
Golang	990	4,06	5,55	18,42	46,18
Nodejs	990	5,18	6,53	32,33	62,15
Python	987	5,83	6,87	13,72	52,68
Golang coldStart	4	212,93	225,82	225,82	225,82
Nodejs coldstart	3	697,7	704,37	704,37	704,37
Python coldStart	6	415,08	448,08	448,08	448,08

Memória utilizada

	Média (MB)	Mínima (MB)	Máxima (MB)
Golang	45,7	42,9	47,7
Nodejs	92,2	89,6	95,4
Python	68,2	65,8	68,7

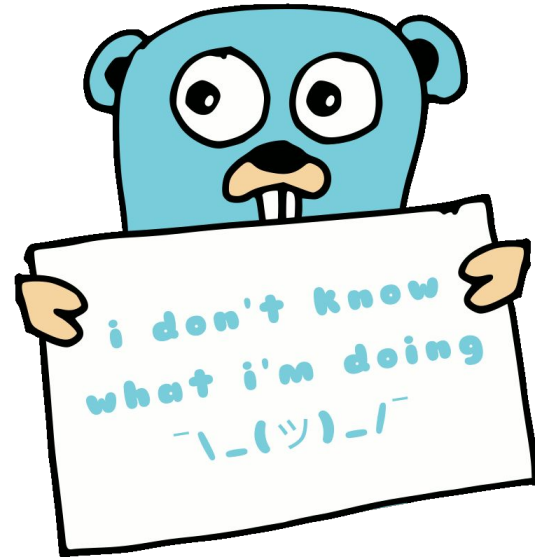
Foi uma boa trazer Go para a empresa?

- Depende, cada um tem uma visão;
- Sempre teremos trade offs.



Opinião

- Serviço com maior número de requisições hoje e estável;
- Zero bugs registrados até o momento;
- Difícil defender escalabilidade em lambda;
- Custo reduzido comparado a uso de outra linguagem no mesmo contexto;
- Plus do melhor -> Maior eficiência e menor latência nos serviços.



Conclusão

- Ótima linguagem para sistemas críticos:
 - Confiabilidade;
 - Escalável;
 - Preço reduzido;
 - Baixa latência / cold start;
- Adequado para aplicações concorrentes e paralelas (nem entramos nesse quesito no bench).
- Entre outros pontos.



Obrigado!

Você tem alguma pergunta?

gprando55@gmail.com
Utilize o QR Code e me add no LinkedIn!



CREDITS: This presentation template was created by [Slidesgo](#), including icons by [Flaticon](#), infographics & images by [Freepik](#) and illustrations by [Stories](#)

Please keep this slide for attribution