

# Golang

---

Guilherme Giacchetto

<https://github.com/guilhermegm>

06/09/2017





Robert Griesemer, Rob Pike and Ken Thompson

# Pq Go?

Performance

Múltiplos cores

## **Concorrência**

Compilado e portátil (gcc-go)

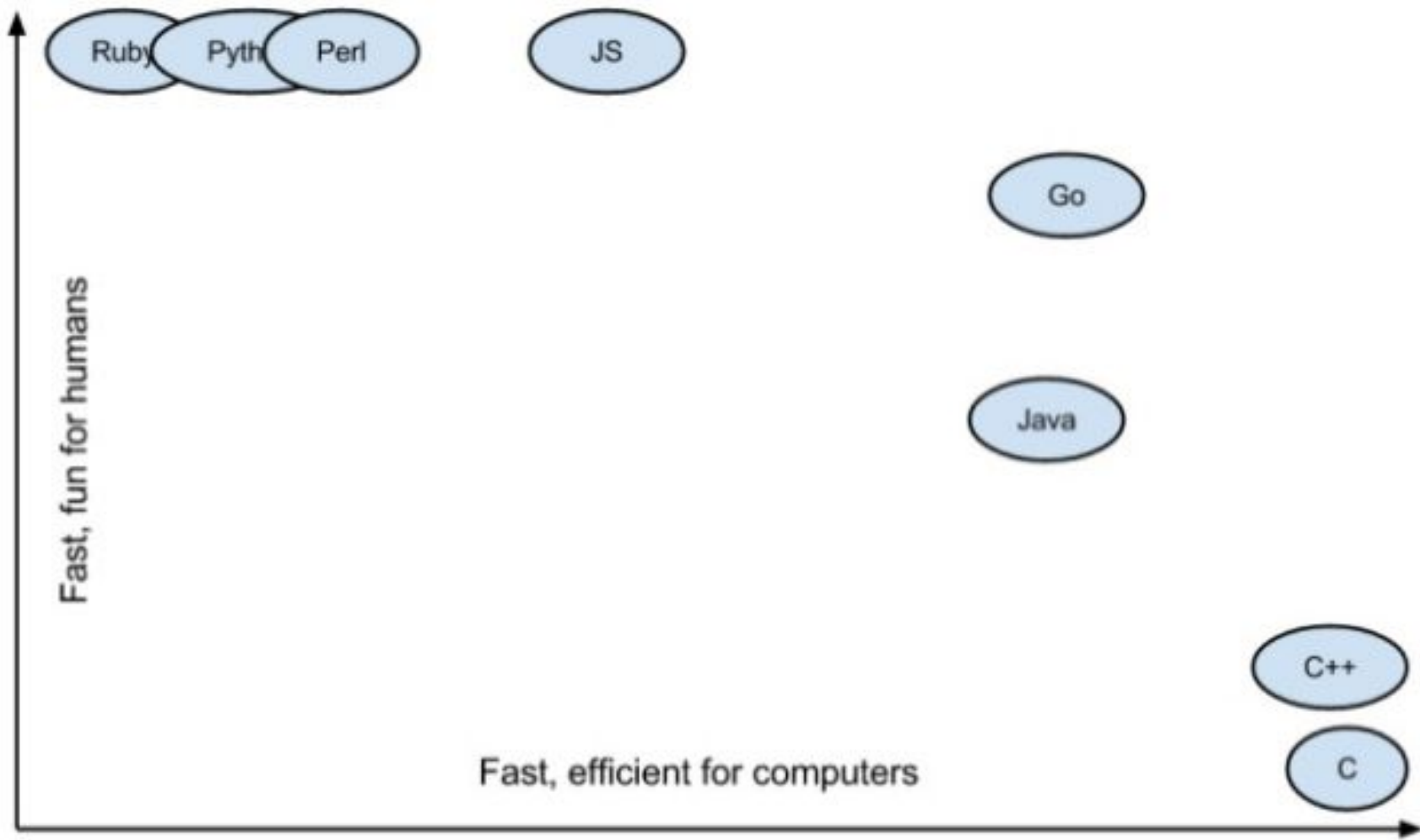
Biblioteca padrão “poderosa”

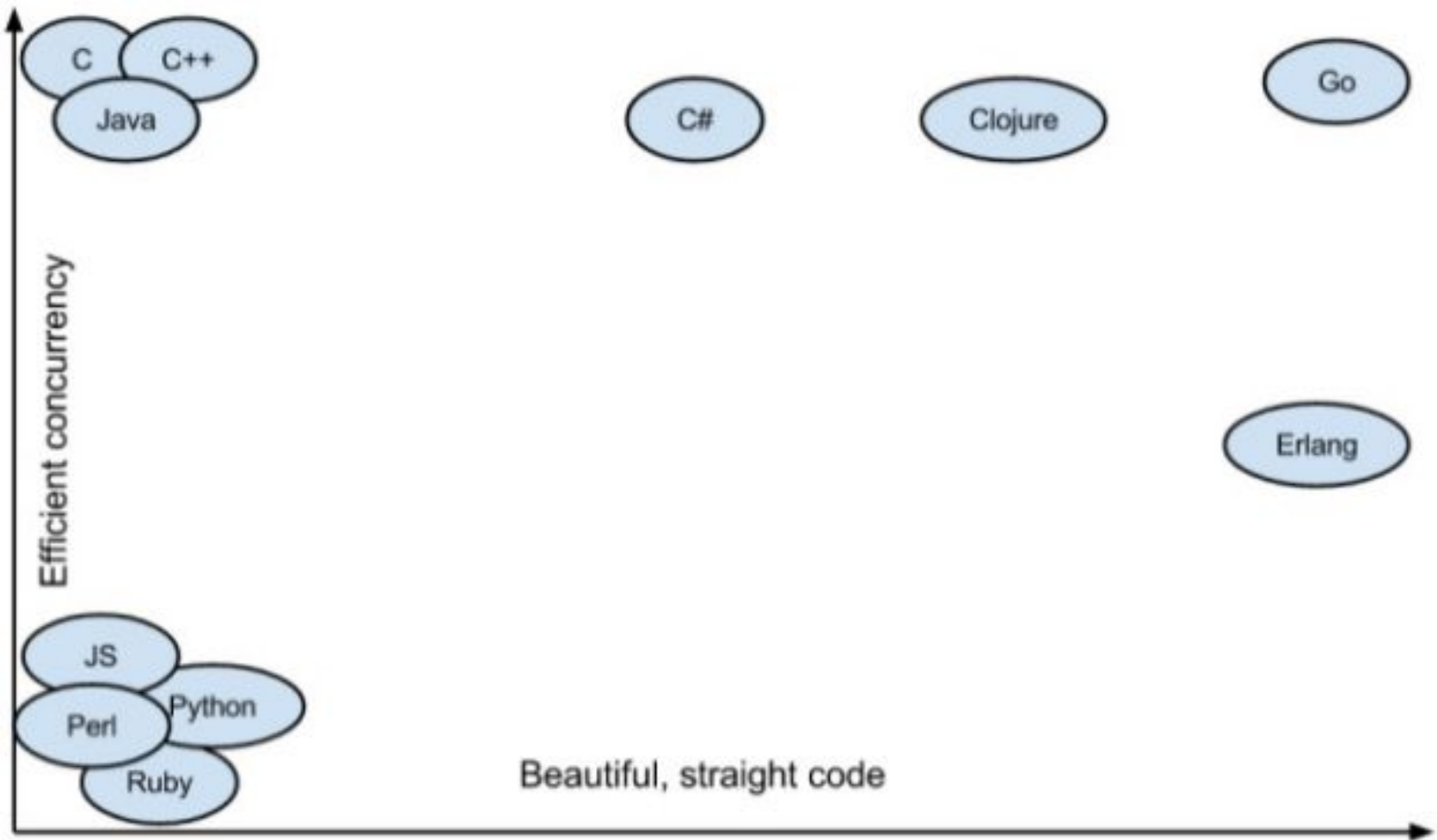
Fortemente tipada

Open Source

# Cross Compilation

<b>\$GOOS</b>	<b>\$GOARCH</b>	
darwin	386	-- 32 bit MacOSX
darwin	amd64	-- 64 bit MacOSX
freebsd	386	
freebsd	amd64	
linux	386	-- 32 bit Linux
linux	amd64	-- 64 bit Linux
linux	arm	-- RISC Linux
netbsd	386	
netbsd	amd64	
openbsd	386	
openbsd	amd64	
plan9	386	
windows	386	-- 32 bit Windows
windows	amd64	-- 64 bit Windows







companies using Go



docker



heroku



Dropbox

<https://github.com/golang/go/wiki/GoUsers>

# Go @ Google

Go is a programming language designed by Google to help solve Google's problems, and Google has big problems.

<https://talks.golang.org/2012/splash.article>



# Cases

<http://blog.parse.com/learn/how-we-moved-our-api-from-ruby-to-go-and-saved-our-sanity/>

<https://eng.uber.com/go-geofence/>

<https://imasters.com.br/linguagens/o-ceu-e-o-limite-na-utilizacao-de-golang/?trace=1519021197&source=single>

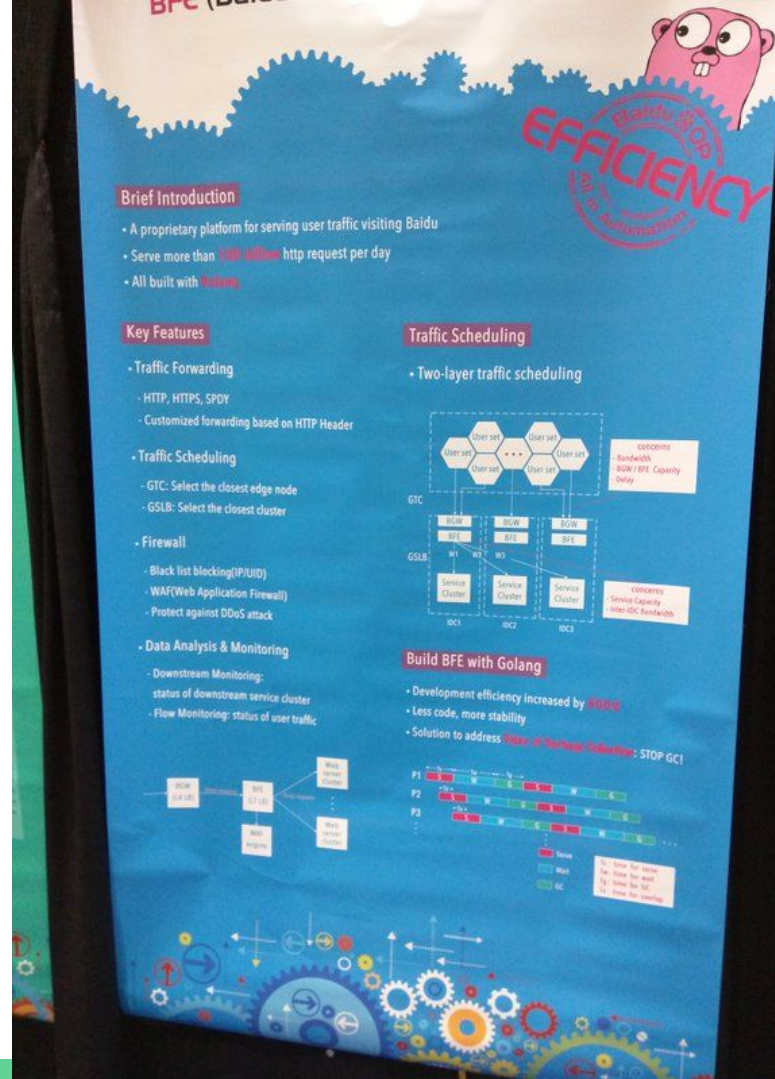
Pool	Grails (4GB Ram e 4 Cores)			Golang (3.75GB Ram e 2 Cores)		
	Equipamentos	Memória (GB)	Cores	Equipamentos	Memória (GB)	Cores
Cache	12	48	48	-	-	-
Leitura	10	40	40	2	7,50	4
Escrita	6	24	24	1	3,75	2
Teste	4	16	16	1	3,75	2
Total	32	128	128	4	15	8

# Cases 2

[https://blog.twitter.com/engineering/en\\_us/a/2015/handling-five-billion-sessions-a-day-in-real-time.html](https://blog.twitter.com/engineering/en_us/a/2015/handling-five-billion-sessions-a-day-in-real-time.html)

[https://www.reddit.com/r/golang/comments/3c86u1/baidus\\_front\\_end\\_http\\_routing\\_stack\\_serves\\_100/](https://www.reddit.com/r/golang/comments/3c86u1/baidus_front_end_http_routing_stack_serves_100/)

<http://marcio.io/2015/07/handling-1-million-requests-per-minute-with-golang/>



The diagram illustrates the Baidu BFE (Baidu Front-End) architecture, which is designed for high efficiency and scalability. It features a blue background with a pink cartoon character in the top right corner and a gear-themed border at the bottom.

**Baidu BFE**

**Brief Introduction**

- A proprietary platform for serving user traffic visiting Baidu
- Serve more than 100 billion http request per day
- All built with **Golang**

**Key Features**

- Traffic Forwarding**
  - HTTP, HTTPS, SPDY
  - Customized forwarding based on HTTP Header
- Traffic Scheduling**
  - GTC: Select the closest edge node
  - GSLB: Select the closest cluster
- Firewall**
  - Black list blocking(IP/UID)
  - WAF(Web Application Firewall)
  - Protect against DDoS attack
- Data Analysis & Monitoring**
  - Downstream Monitoring: status of downstream service cluster
  - Flow Monitoring: status of user traffic

**Traffic Scheduling**

- Two-layer traffic scheduling

The diagram shows a two-layer traffic scheduling architecture. The top layer, labeled GTC, consists of multiple "User set" boxes. The bottom layer, labeled GSLB, consists of multiple "Service Cluster" boxes. The GTC layer is connected to the GSLB layer via "BFE" boxes. The GSLB layer is connected to the "Service Cluster" boxes via "BFE" boxes. The diagram also shows "concerns" for both layers: GTC concerns Bandwidth, B/W / BFE Capacity, and Delay; GSLB concerns Service Capacity and User IDC Bandwidth.

**Build BFE with Golang**

- Development efficiency increased by **400%**
- Less code, more stability
- Solution to address **issues of language inhibitors: STOP GC!**

The diagram also includes a timeline showing the development of BFE with Golang. It shows three parallel timelines for P1, P2, and P3. P1 shows a timeline from T0 to T1, with a red bar indicating the development period. P2 shows a timeline from T0 to T2, with a red bar indicating the development period. P3 shows a timeline from T0 to T3, with a red bar indicating the development period. The diagram also includes a legend for the timeline: Red bar: Development, Green bar: Production, Blue bar: Testing, Yellow bar: Deployment.

# Install Go

```
$ tar -C /usr/local -xzf go$VERSION.$OS-$ARCH.tar.gz
```

```
$ export PATH=$PATH:/usr/local/go/bin
```

# Primeiro Programa

\$ touch hello.go

```
package main

import "fmt"

func main() {
    fmt.Printf("Hello, world.\n")
}
```

# Instalando um pacote

```
$ go install github.com/user/hello
```

# Run e Build

\$ go run hello.go

\$ go build hello.go

\$ ./hello

## if

```
if x > 0 {  
    return y  
} else {  
    return x  
}
```

## Maps

```
var m map[string]int  
monthdays := map[string]int{  
    "Jan": 31, "Feb": 28, "Mar": 31,  
    "Apr": 30, "May": 31, "Jun": 30,  
    "Jul": 31, "Aug": 31, "Sep": 30,  
    "Oct": 31, "Nov": 30, "Dec": 31 }
```

## switch

```
switch {  
case a > b:  
    return true  
case a < b:  
    return false  
}
```

```
switch c {  
case '+', '-', '*', '/':  
    return true  
}  
return false
```

## Arrays

```
var arr [3]int  
arr[0] = 1  
arr[1] = 2  
arr[2] = 3  
  
arr := [3]int{1, 2, 3}  
arr := [...]int{1, 2, 3}
```

## for

```
for {  
}  
  
for err != nil {  
}  
  
for i := 0; i < 10; i++ {  
}
```

# Unit Tests

\*\_test.go

\$ go test

<https://github.com/shekhargulati/52-technologies-in-2016/blob/master/29-go-unit-testing/README.md>



# Code Coverage

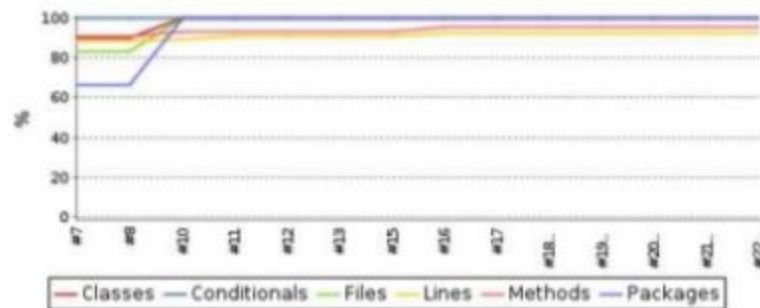
\$ go test

-coverprofile=coverage.out

## Code Coverage

### Cobertura Coverage Report

Trend



### Project Coverage summary

Name	Packages	Files	Classes
Cobertura Coverage Report	100% <div>3/3</div>	100% <div>6/6</div>	100% <div>10/10</div>

### Coverage Breakdown by Package

Name	Files	Classes
<a href="#">github.com/go-gas/sessions</a>	100% <div>3/3</div>	100% <div>6/6</div>
<a href="#">github.com/go-gas/sessions/memory</a>	100% <div>2/2</div>	100% <div>3/3</div>
<a href="#">github.com/go-gas/sessions/tests</a>	100% <div>1/1</div>	100% <div>1/1</div>

# Docs built-in

## func MatchString

```
func MatchString(pattern string, s string) (matched bool, err error)
```

MatchString checks whether a textual regular expression matches a string. More complicated queries need to use Compile and the full Regexp interface.

### ▼ Example

```
package main

import (
    "fmt"
    "regexp"
)

func main() {
    matched, err := regexp.MatchString("foo.*", "seafood")
    fmt.Println(matched, err)
    matched, err = regexp.MatchString("bar.*", "scafood")
    fmt.Println(matched, err)
    matched, err = regexp.MatchString("a{b", "seafood")
    fmt.Println(matched, err)
}
```

```
true <nil>
false <nil>
false error parsing regexp: missing closing ): `a{b`
```

[Run](#)[Format](#)[Share](#)

# Concorrência

[https://play.golang.org/p/6PHXHha\\_Uv](https://play.golang.org/p/6PHXHha_Uv)

# Concorrência 2

Sincronização (`sync.WaitGroup`)

GOMAXPROCS

Race Condition

`sync/MUTEX`

`sync/Atomic`

**Channel**