The State of Go

Where we are on February 2018

Francesc Campoy
VP of Developer Relations at source{d}

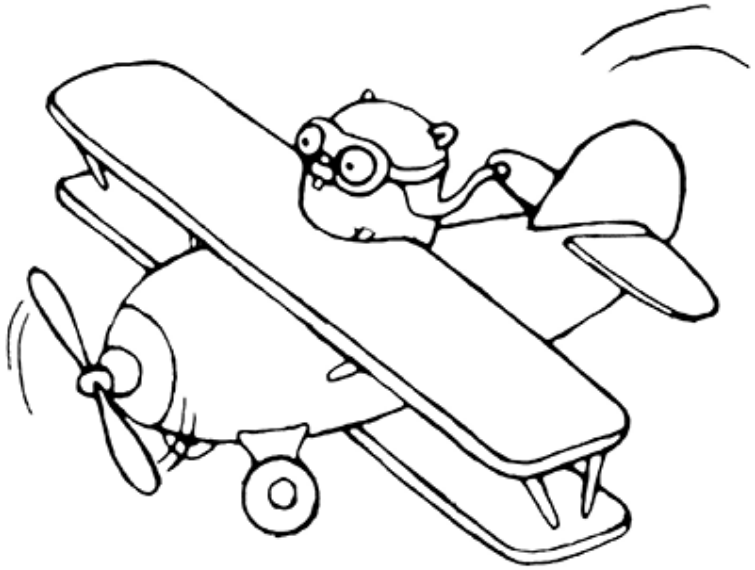
Time flies

Go 1.8 is one year old (Happy Birthday!)

Go 1.9 is already 6 months old!

Go 1.10rc1 was released on January 25th.

Go 1.10 is about ☝ to be released!



Notes

The slides are already available on campoy.cat/l/sog110

Most of the code examples won't run except locally and using Go 1.10.

The playground still runs Go 1.9.

👉 do not send issues about the slides not running correctly online!

Agenda

Changes since Go 1.9:

- The Language
- The Ports
- The Tooling
- The Standard Library
- The Performance
- The Community

Changes To The Language

Changes To The Language

source

Ports

New Ports

source

Notes On Existing Ports

- FreeBSD: requires FreeBSD 10.3 or later
- NetBSD: works but requires NetBSD 8 ... which is not released yet
- OpenBSD: next version will require OpenBSD 6.2
- OS X: next version will require OS X 10.10 Yosemite
- Windows: next version will require Windows 7 (no more XP or Vista)
- 32-bits MIPS have now a new GOMIPS variable (**hardfloat** | `softfloat`)

One More Note On Existing Ports

It's rare that I laugh out loud while reading GitHub issues.

> And even after Go 1.12 comes out, you can keep using Go 1.10, we just won't fix bugs in it. But if you're happy with it (or Go 1.9 or whatever version), great. You won't get security fixes, but if you are running XP you're not worried about that.

😄 3

Changes To The Tooling

Changes To The Tooling

In two words: easier and faster.

Easier set-up

`GOPATH` became optional in Go 1.8.

`GOROOT` is now optional too, deduced from the binary path.

A new variable `GOTMPDIR` was added to control where temporary files are created.

Faster tools via caching

- `go install` now caches the result of compiled packages.
- `go install` and `go build` are **much** faster in general as a result
- you won't need `go build -i` anymore!

It seems the pkg directory might eventually disappear!

Testing

Also caches results, everything is faster

```
➜  go test strings
ok      strings    (cached)
```

In order to bypass the cachee use `-count=1`

```
➜  go test -count=1 strings
ok      strings    0.295s
```

Also runs `vet`, some of your tests might fail.

Also:

- `coverprofile` can be done over many tests too
- new `-failfast` and `-json` flags

A Small Detour

Three-Index Slicing

Did you know you can use three values for slicing?

```
text := []byte("Hello FOSDEM!")
fmt.Printf("text:  %s", desc(text))

hello := text[0:5]
fmt.Printf("hello: %s", desc(hello))

hello = append(hello, '#')
fmt.Printf("hello: %s", desc(hello))

fmt.Printf("text:  %s", desc(text))
```

Run

Three-Index Slicing (cont.)

You can control the capacity of the resulting slice.

```
text := []byte("Hello FOSDEM!")
fmt.Printf("text:  %s", desc(text))

hello := text[0:5:5]
fmt.Printf("hello: %s", desc(hello))

hello = append(hello, '#')
fmt.Printf("hello: %s", desc(hello))

fmt.Printf("text:  %s", desc(text))
```

Run

gofmt

Small change in formatting of three-index slicing expressions.

Before:

```
a[i : j:k]
```

Now:

```
a[i : j : k]
```

This might break some of your CI tests (it broke some of mine).

Changes To The Standard Library

Changes To The Standard Library

No new packages with Go 1.10

Trivia: Do you remember which new package was added with Go 1.9?

Changes to bytes

`Fields`, `FieldsFunc`, `Split`, and `SplitAfter` limit the capacity of the returned slices.

```
    text := []byte("Hello FOSDEM!")
    fmt.Printf("text:  %s", desc(text))

    hello := bytes.Fields(text)[0]
    fmt.Printf("hello: %s", desc(hello))

    hello = append(hello, '#')
    fmt.Printf("hello: %s", desc(hello))

    fmt.Printf("text:  %s", desc(text))
```

Run

playground

Changes to flags

This is minor, but I am very happy about it!

```
    stuff := flag.Int("s", 0, "some other stuff\nit's long to explain")
    z := flag.Int("z", 42, "some number")
    flag.Parse()
```

Run

Before 😕

```
-s int
        some other stuff
it's long to explain
-z int
        some number (default 42)
```

Now 😎

```
-s int
        some other stuff
        it's long to explain
-z int
        some number (default 42)
```

Changes to go/doc

For a type T, functions returning slices of T, *T, or **T are now linked to T.

Those functions now appear in the Funcs list of the type, not the package.

Example:

```
package things

// Thing is stuff.
type Thing struct{}

// NewThing returns a new thing.
func NewThing() *Thing { return nil }

// ManyThings returns many new things.
func ManyThings() []Thing { return nil }
```

Changes to go/doc (cont.)

Before 😕

```
package things // import "github.com/campoy/talks/go1.10/things"

func ManyThings() []Thing
type Thing struct{}
    func NewThing() *Thing
```

Now 😎

```
package things // import "github.com/campoy/talks/go1.10/things"

type Thing struct{}
    func ManyThings() []Thing
    func NewThing() *Thing
```

Changes to text/template

New `{{break}}` and `{{continue}}` for `{{range}}`.

```
var tmpl = template.Must(template.New("example").Funcs(template.FuncMap{
    "even": func(x int) bool { return x%2 == 0 },
}).Parse(`
{{ range . }}
    {{ . }}
    {{ if even . -}}
        even
        {{ continue }}
    {{ end -}}
    odd
    {{ if eq . 5 }}
        {{ break }}
    {{ end }}
{{ end }}
`))
```

Run

Note: Interestingly, this is not implemented in the `html` package.

strings

I'm sure you've written this kind of code before.

```
var buf bytes.Buffer
fmt.Fprintln(&buf, "Hello, FOSDEM gophers!")
fmt.Printf(buf.String())
```

Run

But there's some issues with it.

`String` creates allocations since it convers `[]byte` to `string`.

There could be a better and simpler way to do this.

```
var b strings.Builder
fmt.Fprintln(&b, "Hello, FOSDEM gophers!")
fmt.Printf(b.String())
```

Run

This uses `unsafe` to avoid copies in the creation of strings.

strings.Builder 👍

When you're creating many strings, it is definitely worth it.

```
    for i := 0; i < 10000; i++ {
        fmt.Fprintf(w, "😊")
        out = w.String()
    }
```

Benchmark results:

```
$ go test -bench=. -benchmem
goos: darwin
goarch: amd64
pkg: github.com/campoy/talks/go1.10/strings
BenchmarkBuffer-4            100          20861915 ns/op        215641272 B/op     10317 allocs/op
BenchmarkBuilder-4          3000            535081 ns/op          153647 B/op        22 allocs/op
PASS
ok      github.com/campoy/talks/go1.10/strings  3.626s
```

strings.Builder 👎

When you're creating many strings, it is definitely worth it.

```
for i := 0; i < 10000; i++ {
    fmt.Fprintf(w, "😊")
    // out = w.String()
}
```

Benchmark results:

```
$ go test -bench=. -benchmem
goos: darwin
goarch: amd64
pkg: github.com/campoy/talks/go1.10/strings
BenchmarkBuffer-4          3000            525691 ns/op          152056 B/op         11 allocs/op
BenchmarkBuilder-4         3000            626132 ns/op          153647 B/op         22 allocs/op
PASS
ok      github.com/campoy/talks/go1.10/strings  4.072s
```

unicode

source

unicode

oh my gopher!

unicode

sure ... why not

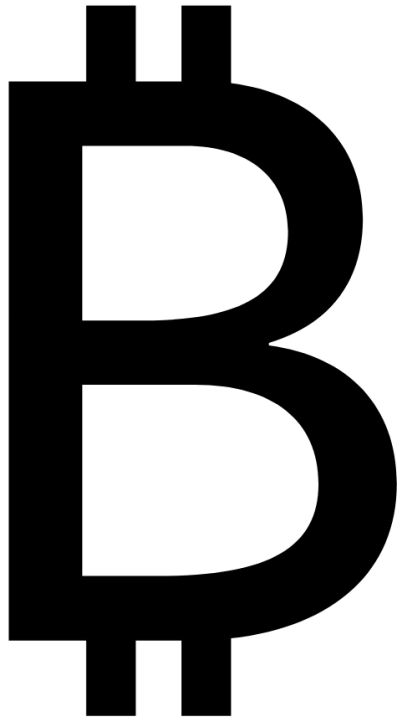
unicode

roar

unicode

mind blown

and the unicode character we all wanted



the character we deserve

Performance Changes

Runtime Performance

After running all the benchmakrks on the standard library on go1.9.3 vs go1.10rc1:

- nothing changed

```
$ benchstat go1.9.txt go1.10.txt  | grep -v "\~"
```

source

Compiler Performance

Compiling the standard library is **10% faster**!

```
$ benchstat go1.9.3.txt go.1.10rc1.txt
name       old time/op       new time/op       delta
Template         234ms ± 4%        231ms ± 4%     ~     (p=0.101 n=10+8)
Unicode          107ms ± 1%        109ms ± 6%     ~     (p=0.211 n=9+10)
GoTypes          742ms ± 2%        744ms ± 2%     ~     (p=0.905 n=9+10)
Compiler         3.50s ± 3%        3.54s ± 5%     ~     (p=0.393 n=10+10)
SSA              6.95s ± 4%        9.04s ± 5%  +29.98%  (p=0.000 n=10+10)
Flate            149ms ± 2%        147ms ± 5%   -1.53%  (p=0.035 n=10+9)
GoParser         189ms ± 3%        183ms ± 3%   -3.44%  (p=0.002 n=9+9)
Reflect          476ms ± 5%        489ms ± 6%   +2.90%  (p=0.043 n=10+10)
Tar              134ms ± 1%        220ms ± 3%  +64.14%  (p=0.000 n=9+10)
XML              258ms ± 6%        266ms ± 6%   +2.90%  (p=0.043 n=10+10)
StdCmd           19.1s ± 1%        17.1s ± 3%  -10.57%  (p=0.000 n=10+10)
```

Following https://golang.org/x/tools/cmd/compilebench.

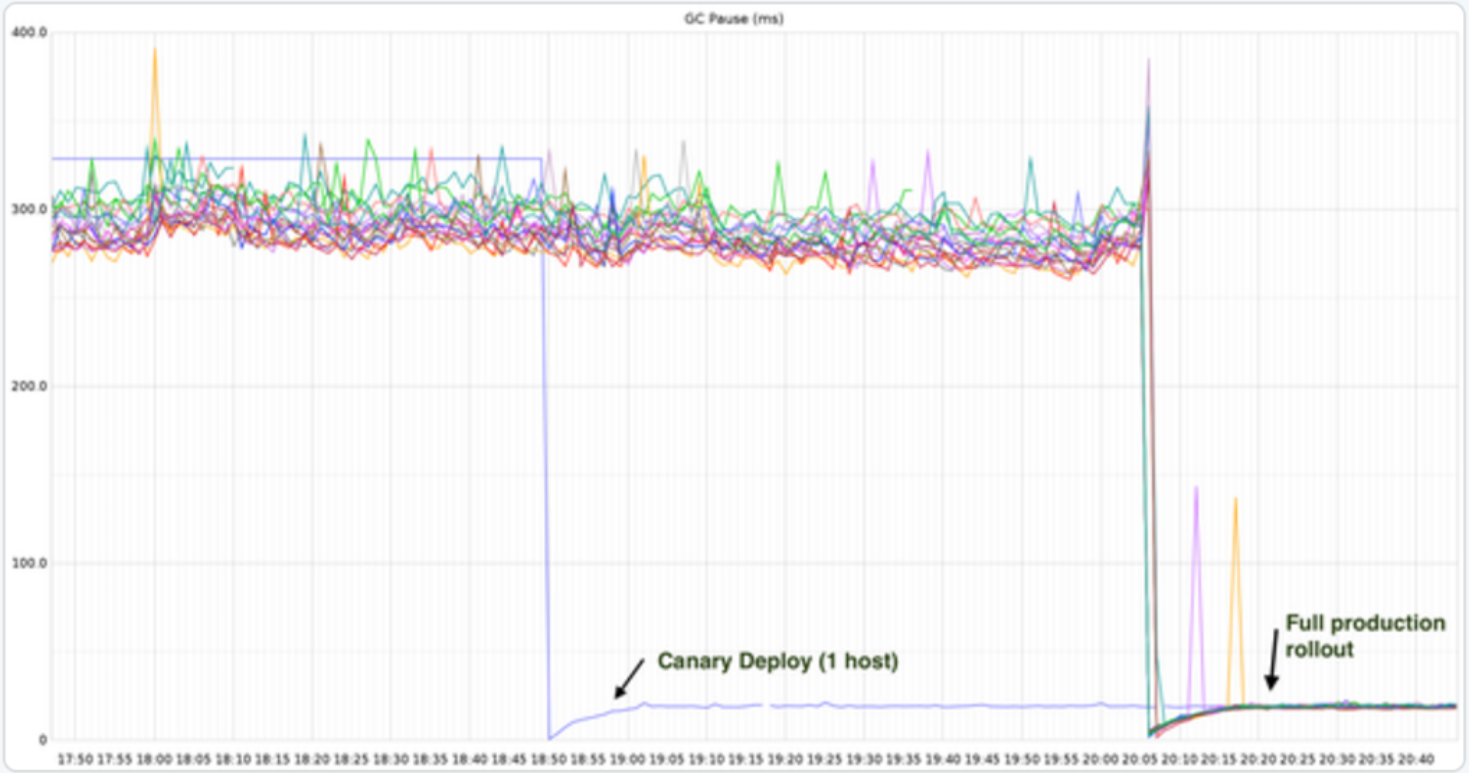
Run on a Google Compute Engine instance with 8 cores.

Garbage Collector History in Tweets

go 1.5

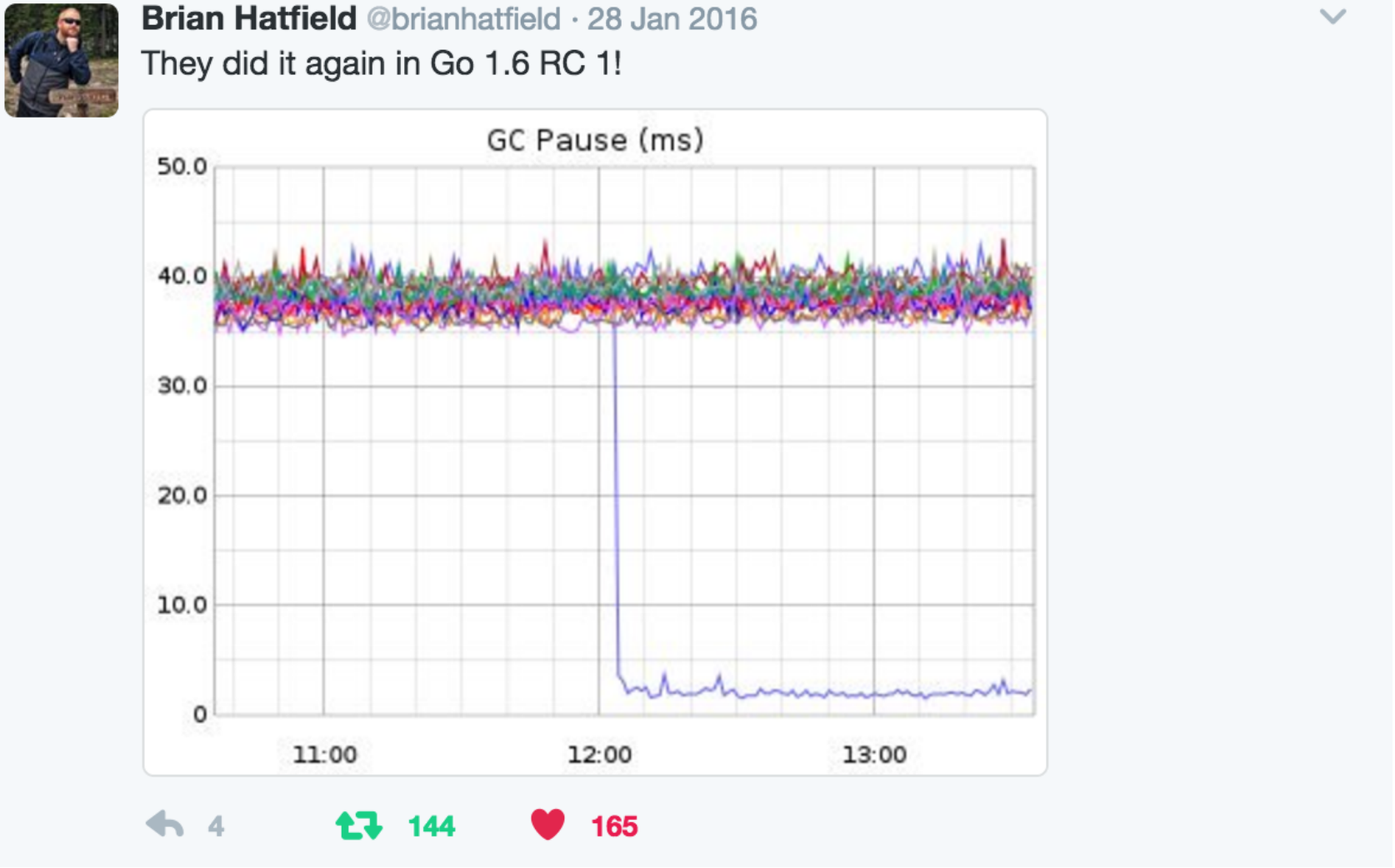
Brian Hatfield @brianhatfield · 19 Aug 2015

Amazing GC pause time improvements in Go 1.5.

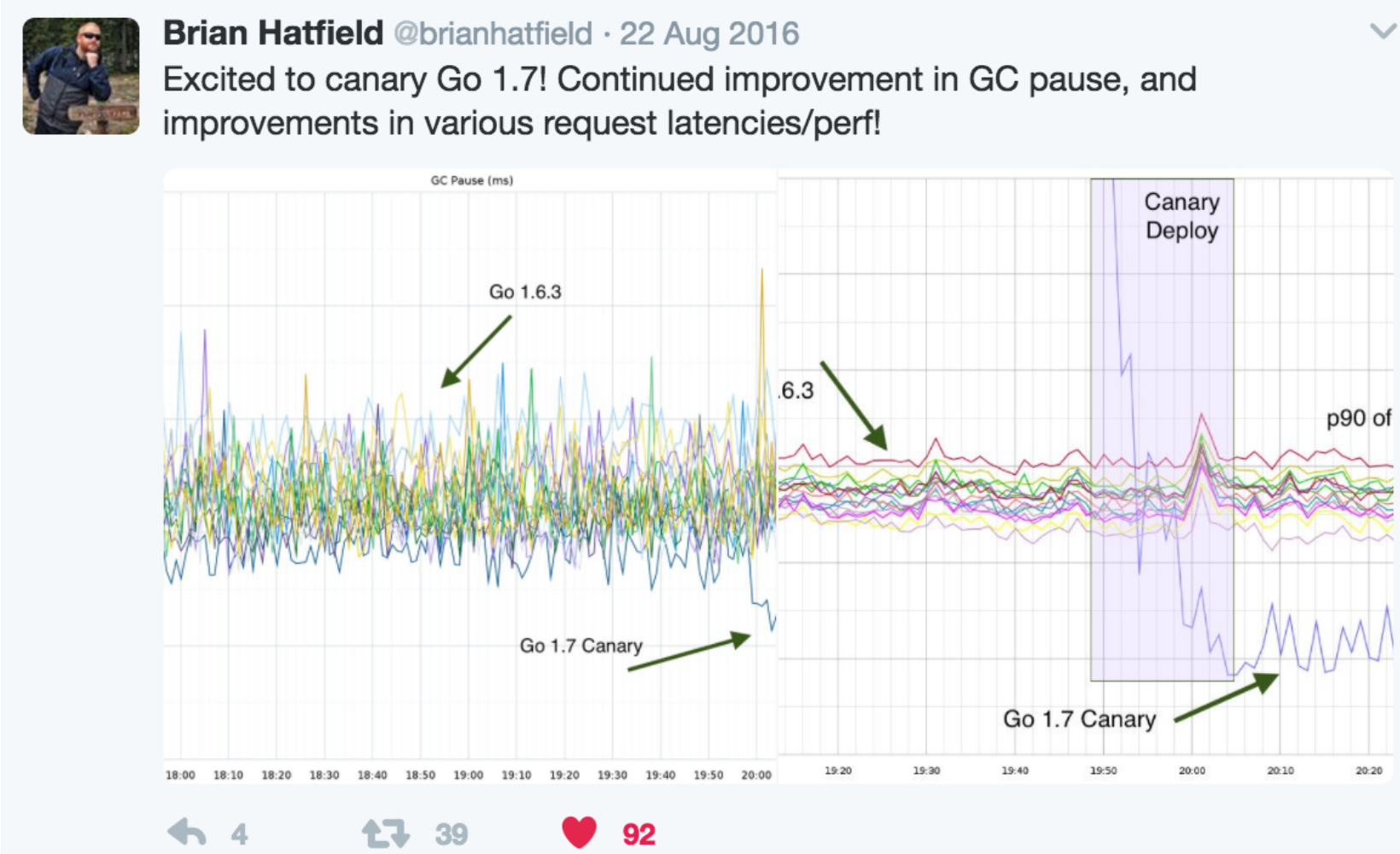


10 227 230

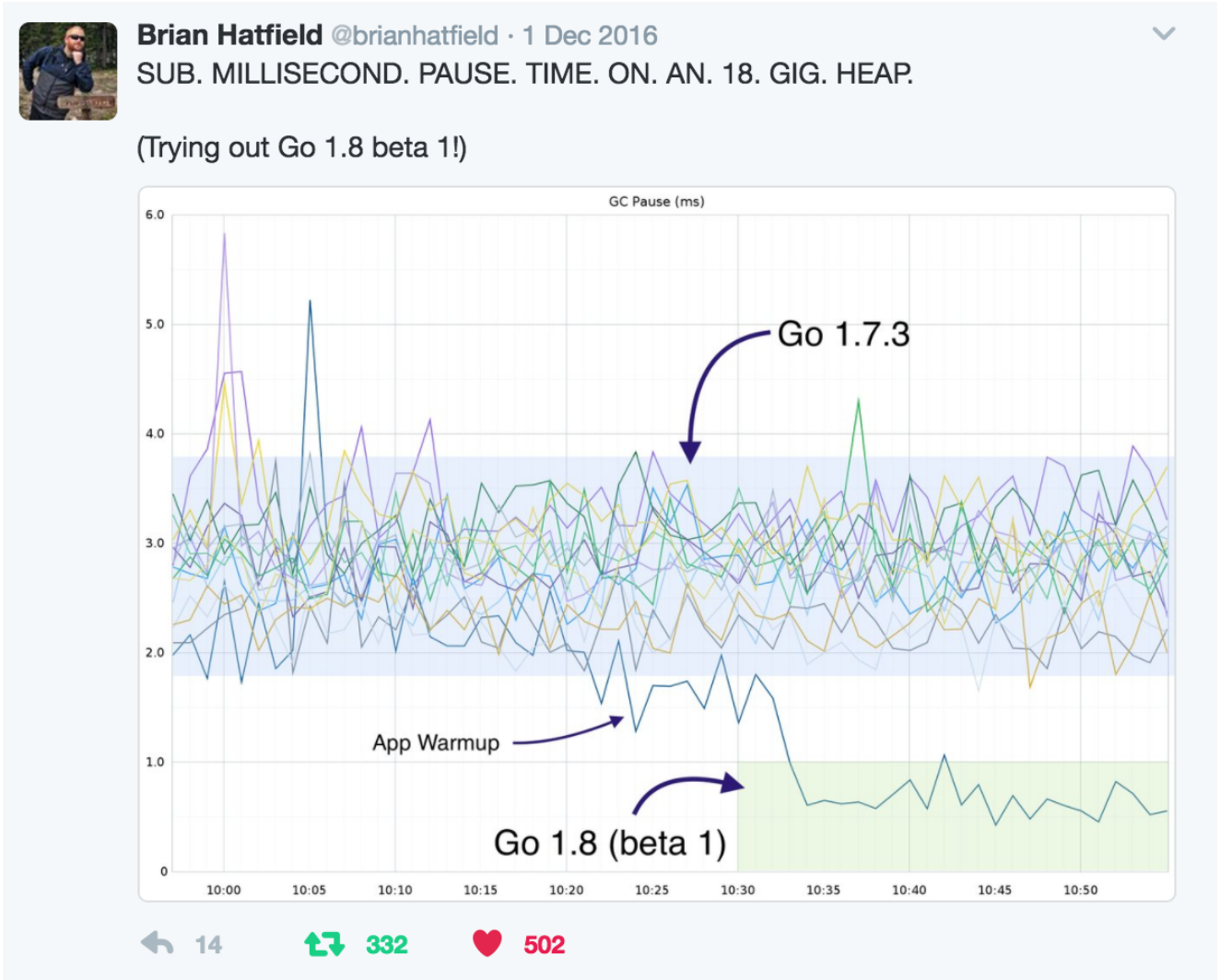
go 1.6


Brian Hatfield @brianhatfield · 28 Jan 2016
They did it again in Go 1.6 RC 1!
GC Pause (ms)
50.0
40.0
30.0
20.0
10.0
0
11:00
12:00
13:00
4
144
165

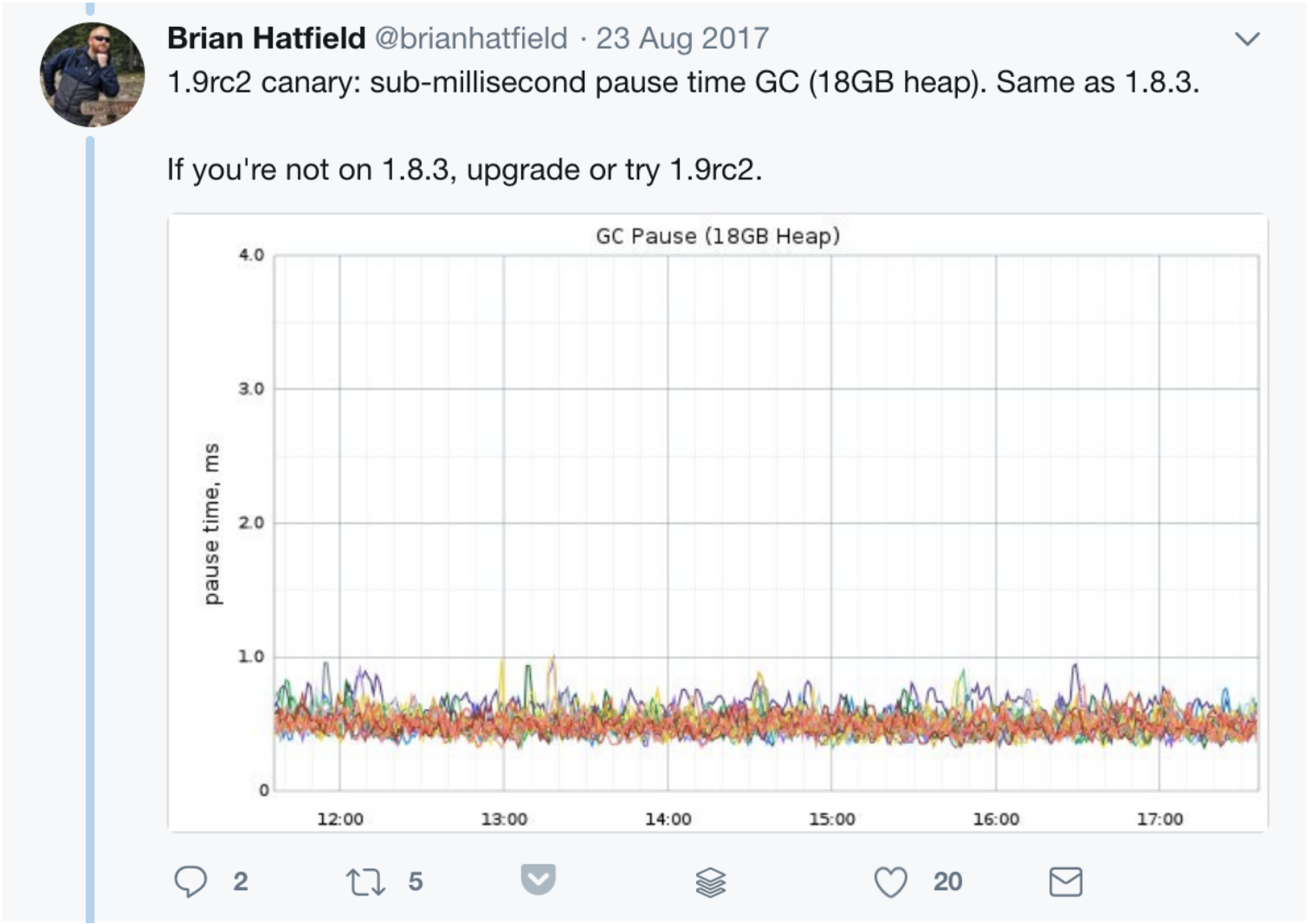
go 1.7


Brian Hatfield @brianhatfield · 22 Aug 2016
Excited to canary Go 1.7! Continued improvement in GC pause, and improvements in various request latencies/perf!
GC Pause (ms)
Go 1.6.3
Go 1.7 Canary
18:00 18:10 18:20 18:30 18:40 18:50 19:00 19:10 19:20 19:30 19:40 19:50 20:00
Canary Deploy
.6.3
p90 of
Go 1.7 Canary
19:20 19:30 19:40 19:50 20:00 20:10 20:20
4 39 92

go 1.8 (beta 1)


Brian Hatfield @brianhatfield · 1 Dec 2016
SUB. MILLISECOND. PAUSE. TIME. ON. AN. 18. GIG. HEAP.
(Trying out Go 1.8 beta 1!)
GC Pause (ms)
6.0
5.0
4.0
3.0
2.0
1.0
0
Go 1.7.3
App Warmup
Go 1.8 (beta 1)
10:00
10:05
10:10
10:15
10:20
10:25
10:30
10:35
10:40
10:45
10:50
14
332
502

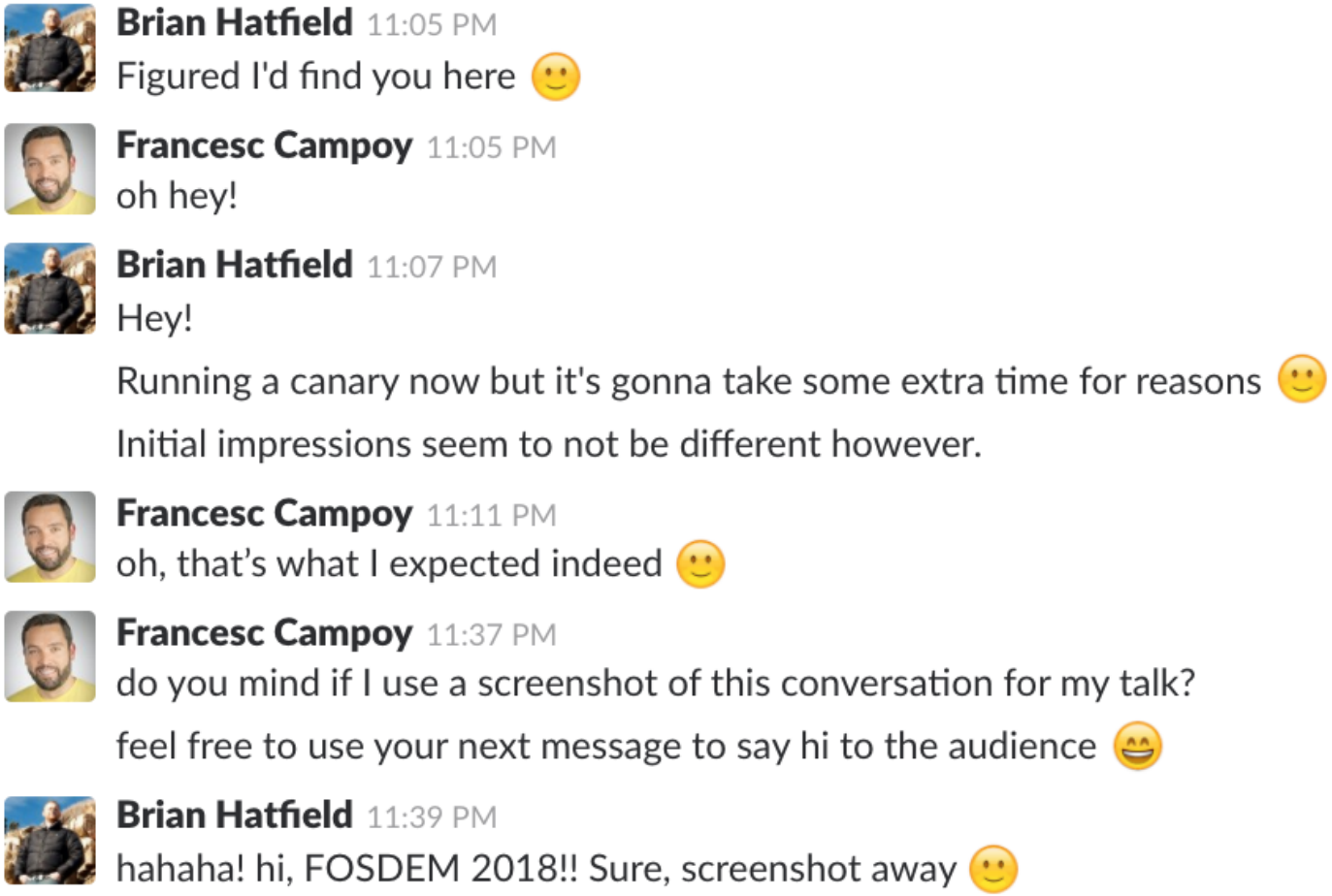
go 1.9 (beta 1)


Brian Hatfield @brianhatfield · 23 Aug 2017
1.9rc2 canary: sub-millisecond pause time GC (18GB heap). Same as 1.8.3.
If you're not on 1.8.3, upgrade or try 1.9rc2.
GC Pause (18GB Heap)
pause time, ms
4.0
3.0
2.0
1.0
0
12:00
13:00
14:00
15:00
16:00
17:00
2
5
20

and finally, go 1.10

and finally, go 1.10

and finally, go 1.10



Brian Hatfield 11:05 PM
Figured I'd find you here 🙂

Francesc Campoy 11:05 PM
oh hey!

Brian Hatfield 11:07 PM
Hey!

Running a canary now but it's gonna take some extra time for reasons 🙂

Initial impressions seem to not be different however.

Francesc Campoy 11:11 PM
oh, that's what I expected indeed 🙂

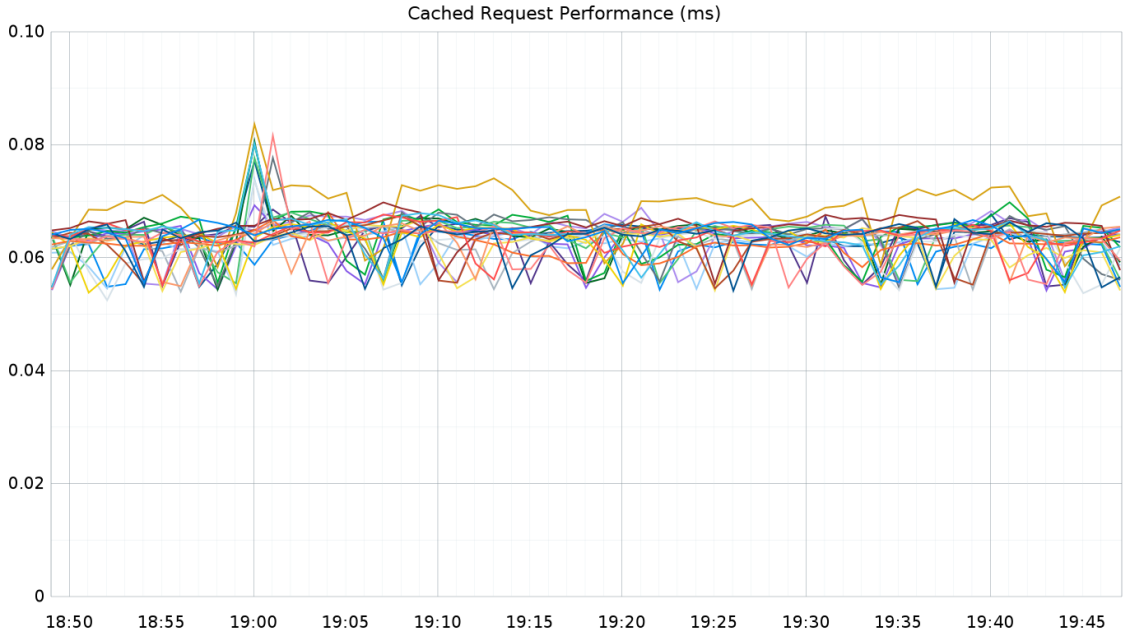
Francesc Campoy 11:37 PM
do you mind if I use a screenshot of this conversation for my talk?

feel free to use your next message to say hi to the audience 😄

Brian Hatfield 11:39 PM
hahaha! hi, FOSDEM 2018!! Sure, screenshot away 🙂

and then this morning ...

and the this morning ...



Brian Hatfield
@brianhatfield

Following

Go 1.10rc1 canary: no significant performance change observations - GC pause, request latency, CPU usage all effectively the same as 1.9.

4:39 PM - 2 Feb 2018

A couple more changes too

archive/tar

In general, the handling of special header formats is significantly improved and expanded.

`FileInfoHeader` has always recorded the Unix UID and GID numbers from its `os.FileInfo` argument (specifically, from the system-dependent information returned by the `FileInfo`'s `Sys` method) in the returned `Header`. Now it also records the user and group names corresponding to those IDs, as well as the major and minor device numbers for device files.

The new `Header.Format` field of type `Format` controls which tar header format the `Writer` uses. The default, as before, is to select the most widely-supported header type that can encode the fields needed by the header (USTAR if possible, or else PAX if possible, or else GNU). The `Reader` sets `Header.Format` for each header it reads.

`Reader` and the `Writer` now support arbitrary PAX records, using the new `Header.PAXRecords` field, a generalization of the existing `Xattrs` field.

The `Reader` no longer insists that the file name or link name in GNU headers be valid UTF-8.

When writing PAX- or GNU-format headers, the `Writer` now includes the `Header.AccessTime` and `Header.ChangeTime` fields (if set). When writing PAX-format headers, the times include sub-second precision.

archive/zip

Go 1.10 adds more complete support for times and character set encodings in ZIP archives.

The original ZIP format used the standard MS-DOS encoding of year, month, day, hour, minute, and second into fields in two 16-bit values. That encoding cannot represent time zones or odd seconds, so multiple extensions have been introduced to allow richer encodings. In Go 1.10, the `Reader` and `Writer` now support the widely-understood Info-Zip extension that encodes the time separately in the 32-bit Unix "seconds since epoch" form. The `FileHeader`'s new `Modified` field of type `time.Time` obsoletes the `ModifiedTime` and `ModifiedDate` fields, which continue to hold the MS-DOS encoding. The `Reader` and `Writer` now adopt the common convention that a ZIP archive storing a time zone-independent Unix time also stores the local time in the MS-DOS field, so that the time zone offset can be inferred. For compatibility, the `ModTime` and `SetModTime` methods behave the same as in earlier releases; new code should use `Modified` directly.

The header for each file in a ZIP archive has a flag bit indicating whether the name and comment fields are encoded as UTF-8, as opposed to a system-specific default encoding. In Go 1.8 and earlier, the `Writer` never set the UTF-8 bit. In Go 1.9, the `Writer` changed to set the UTF-8 bit almost always. This broke the creation of ZIP archives containing Shift-JIS file names. In Go 1.10, the `Writer` now sets the UTF-8 bit only when both the name and the comment field are valid UTF-8 and at least one is non-ASCII. Because non-ASCII encodings very rarely look like valid UTF-8, the new heuristic should be correct nearly all the time. Setting a `FileHeader`'s new `NonUTF8` field to true disables the heuristic entirely for that file.

The `Writer` also now supports setting the end-of-central-directory record's comment field, by calling the `Writer`'s new `SetComment` method.

bufio

The new `Reader.Size` and `Writer.Size` methods report the `Reader` or `Writer`'s underlying buffer size.

bytes

The `Fields`, `FieldsFunc`, `Split`, and `SplitAfter` functions have always returned subslices of their inputs. Go 1.10 changes each returned subslice to have capacity equal to its length, so that appending to one cannot overwrite adjacent data in the original input.

crypto/cipher

`NewOFB` now panics if given an initialization vector of incorrect length, like the other constructors in the package always have. (Previously it returned a nil `Stream` implementation.)

crypto/tls

The TLS server now advertises support for SHA-512 signatures when using TLS 1.2. The server already supported the signatures, but some clients would not select them unless explicitly advertised.

crypto/x509

`Certificate.Verify` now enforces the name constraints for all names contained in the certificate, not just the one name that a client has asked about. Extended key usage restrictions are similarly now checked all at once. As a result, after a certificate has been validated, now it can be trusted in its entirety. It is no longer necessary to revalidate the certificate for each additional name or key usage.

Parsed certificates also now report URI names and IP, email, and URI constraints, using the new `Certificate` fields `URIs`, `PermittedIPRanges`, `ExcludedIPRanges`, `PermittedEmailAddresses`, `ExcludedEmailAddresses`, `PermittedURIDomains`, and `ExcludedURIDomains`.

The new `MarshalPKCS1PublicKey` and `ParsePKCS1PublicKey` functions convert an RSA public key to and from PKCS#1-encoded form.

The new `MarshalPKCS8PrivateKey` function converts a private key to PKCS#8-encoded form. (`ParsePKCS8PrivateKey` has existed since Go 1.)

crypto/x509/pkix

`Name` now implements a `String` method that formats the X.509 distinguished name in the standard RFC 2253 format.

Changes To The Community

Women Who Go

26 chapters already - 10 more than last year! www.womenwhogo.org

Women Who Go Leaders

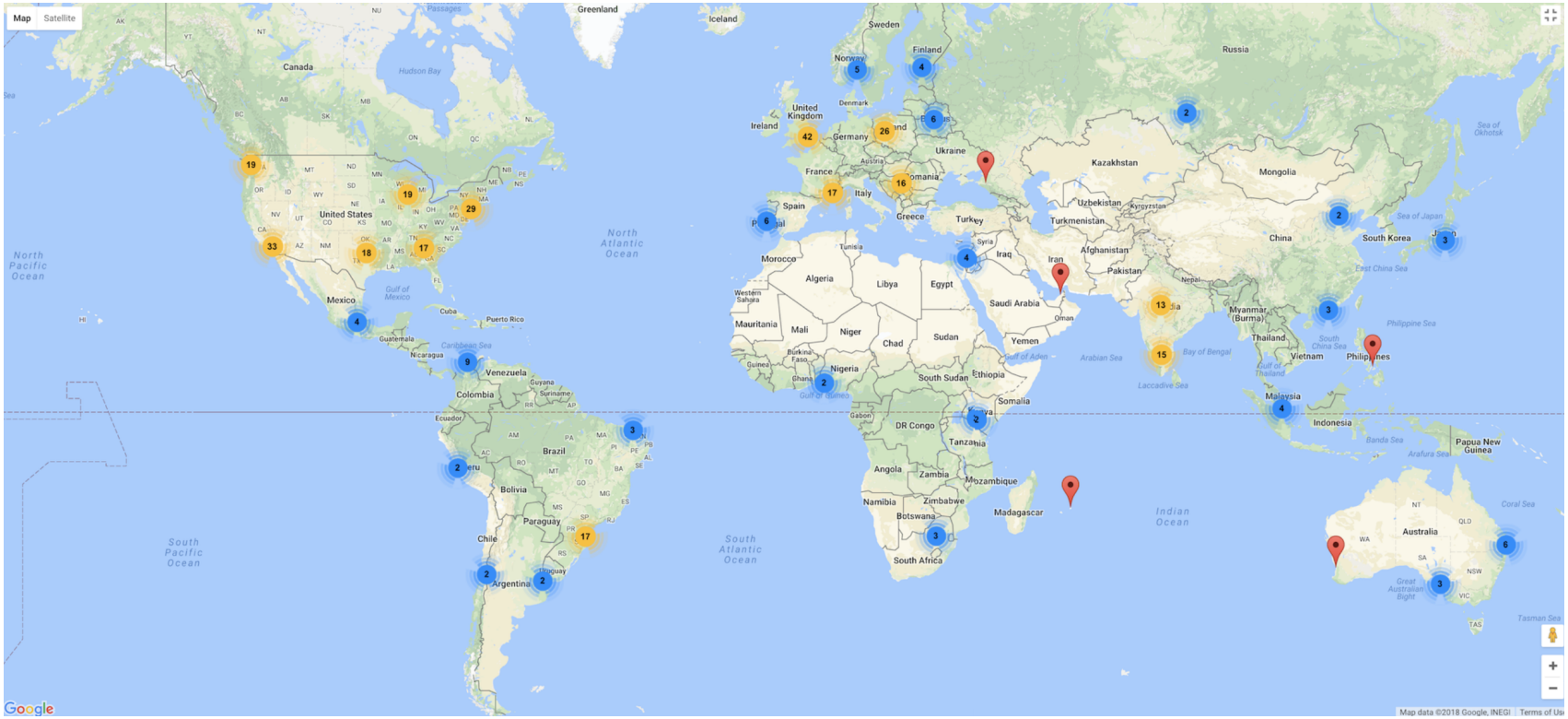
CEO / Global Visionary & Lead
Maartje Eyskens

CFO / Financial Director
Verónica López

Head of New Chapters
Daniela Petruzalek

Head of Support
Carolyn Van Slyck

Go meetups



Gophers all around the world! (367 meetups on go-meetups.appspot.com)

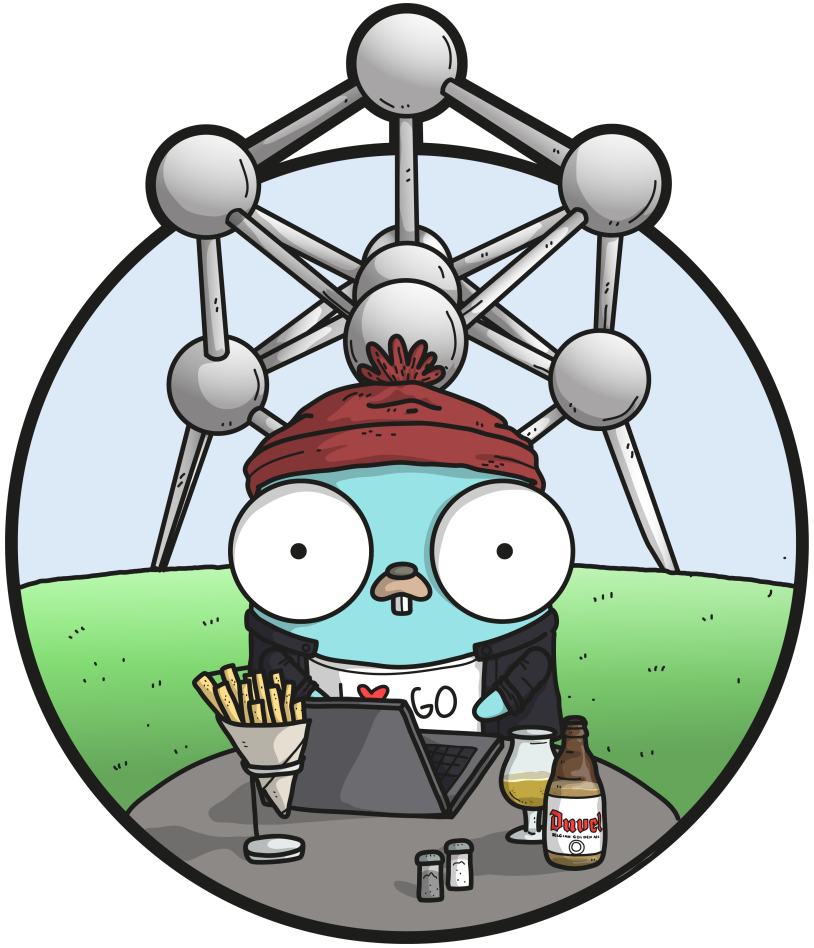
Conferences:

- Go Devroom FOSDEM Today and here! 🎉
- GopherCon India - March in Pune, India
- GopherCon Russia - March in Moscow, Russia
- GoSF - March in San Francisco, USA
- GothamGo - April in New York, USA
- GopherCon SG - May in Singapore
- GopherCon Europe - June in Reykjavik, Iceland
- GopherCon Denver - August in Denver, USA
- GopherCon Brasil - September in Florianópolis, Brazil
- GoLab - October in Florence, Italy
- dotGo - March 2019 in Paris, France

Schedule

	The State of Go *What's new in Go 1.10*	Francesc Campoy	10:30	11:00
	Advanced Go debugging with Delve	Derek Parker	11:00	11:30
	Testing and Automation in the Era of Containers *(with Go)*	Verónica López	11:30	12:00
	Networking deepdive *From net.Dial to gRPC*	Michael Hausenblas	12:00	12:30
	Upspin and a future of the Internet *My vision of Rob Pike's Upspin as a basis for a decentralized Internet*	Gildas Chabot	12:30	13:00
	Dep Deep Dive!	Sam Boyer	13:00	13:30
	Networking Swiss Army Knife for Go	Roman Mohr	13:30	14:00
	The case for interface{} *When and how to use empty interface*	Sam Whited	14:00	14:30
	Google's approach to distributed systems observability for Go	Jaana Dogan (JBD)	14:30	15:00
	Creating GopherJS Apps with gRPC-Web	Johan Brandhorst	15:00	15:30
	Building and testing a distributed data store in Go	Matt Bostock	15:30	16:00
	Computer Vision Using Go And OpenCV	Ron Evans	16:00	16:30
	Make your Go go faster! *Optimising performance through reducing memory allocations*	Bryan Boreham	16:30	17:00
	Distributing DevOps tools using GoLang and Containers, for Fun and Profit!	Lucy Davinhart	17:00	17:30
	AMENDMENT DNA sequencing performance in Go, C++, and Java	Pascal Costanza	17:30	18:00
	Go Lightning Talks *Come speak!*	Francesc Campoy, Maartje Eyskens	18:00	19:00

Enjoy the rest of the day!



Gopher by the amazing Ashley McNamara

Thank you

Francesc Campoy
VP of Developer Relations at source{d}
@francesc
campoy@golang.org
https://sourced.tech