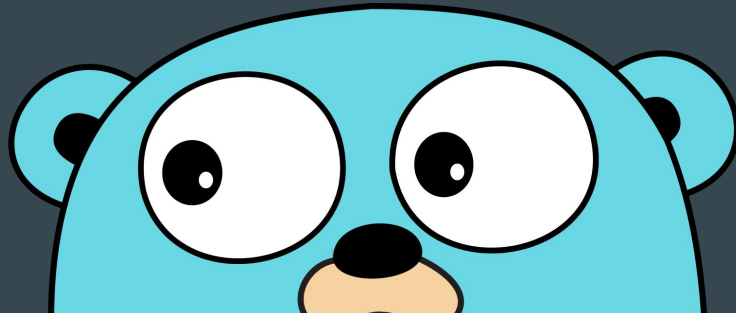


Go After 4 Years in Production

...

Travis Reeder - Co-Founder and CTO of 



Part 1: In the Beginning

6 Years Ago

We started a consulting company.

Built software for smart hardware companies (IoT).

Had to collect, process and report on large, constant streams of data.

DIY job processing.

5 Years Ago

We built our own multi-tenant, job processing system on the cloud.

It enabled us to develop projects faster and with less maintenance.

Good for us, good for our customers.

4 Years Ago


“There must be other developers that have the same problem???”

So we released our new service to the public:



Houston, We Have a Problem

We wrote it in Ruby.

Ruby was 

Ruby for our API

We tried to keep CPU usage < 50% across our servers.

When it increased beyond that, we'd launch more servers to keep it around 50%.

This is fine if you don't mind spending money.

What Amazon thought of us



MAKE GIFS AT GIFSOUP.COM

The Bigger Problem - Traffic Spikes

Traffic spikes caused CPU levels to spike.

At some threshold above 50% CPU, a server would spike up to 100%.

100% == unresponsive

Load balancer removes server from pool.

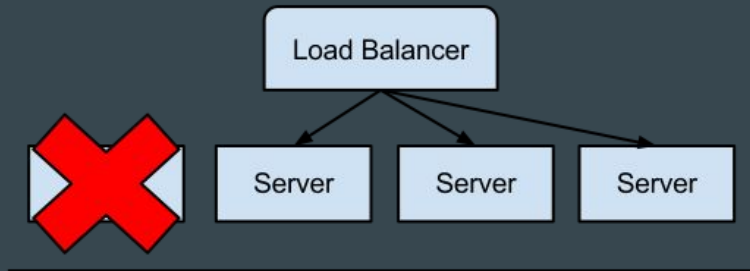
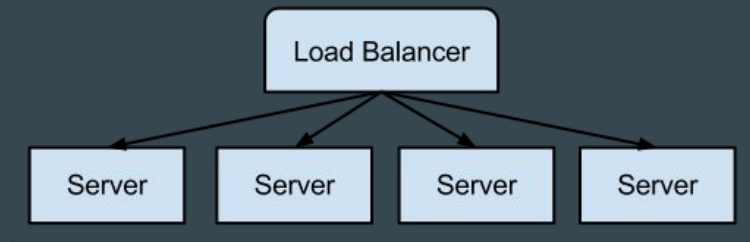
Load distributed to remaining servers.

Dominoes

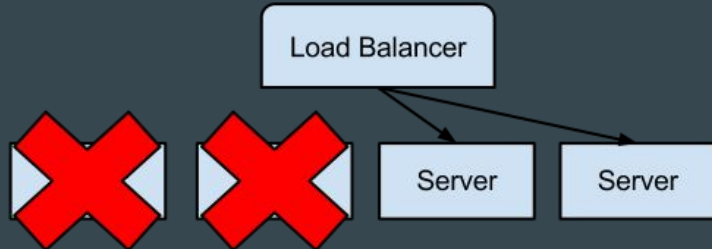
More load on remaining machines => more servers unresponsive => more servers taken offline. => repeat...

Colossal Clusterf**k

Colossal Cluster**k Visualized



One server goes down which puts more load on the remaining servers.



Another goes down which puts more load on the remaining servers.

When your API goes down



What To Do? What To Do?

- 1) Spend more money for extra capacity
- 2) Rewrite it

Part 2: Choosing Go

Choosing a Language

We looked at other scripting languages with better performance than Ruby (wasn't hard).

We looked at Java and derivatives like Scala and Clojure

We looked at Go.

Why We Chose Go

- Concurrency being a fundamental part of the language
- Standard library had almost everything we needed
- It's terse
- It's compiled
- It compiles fast
- It runs fast
- Google is behind it
- It's fun (like Ruby)

Concurrency in Go

2 Main Concurrency Features:

Goroutines and Channels

Goroutines

A goroutine is like a super light weight thread.

Running hundreds of thousands of goroutines is no problem (unlike threads).

You don't need to think much about them (unlike threads).

Goroutines are multiplexed onto threads behind the scenes.

Goroutines

Easy to use:

```
func hello(name string) {  
    fmt.Println("Hello", name)  
}
```

To run function in a goroutine:

```
go hello("Travis")
```

Channels

Channels allow you to communicate between goroutines.

Without having to worry about synchronization (locks, deadlocks, etc).

Channels

```
c := make(chan string)
go hello("Travis", c)
for i := 0; i < 5; i++ {
    fmt.Printf("You say: %q\n", <-c) // receive
}
```

```
func hello(name string, c chan string) {
    for i := 0; ; i++ {
        c <- fmt.Sprintf("Hello %s %d", name, i) // send
    }
}
```

Quote - Rob Pike - 2011

“We realized that the kind of software we build at Google is not always served well by the languages we had available. Robert Griesemer, Ken Thompson, and myself decided to make a language that would be very good for writing the kinds of programs we write at Google.”

Why not X?

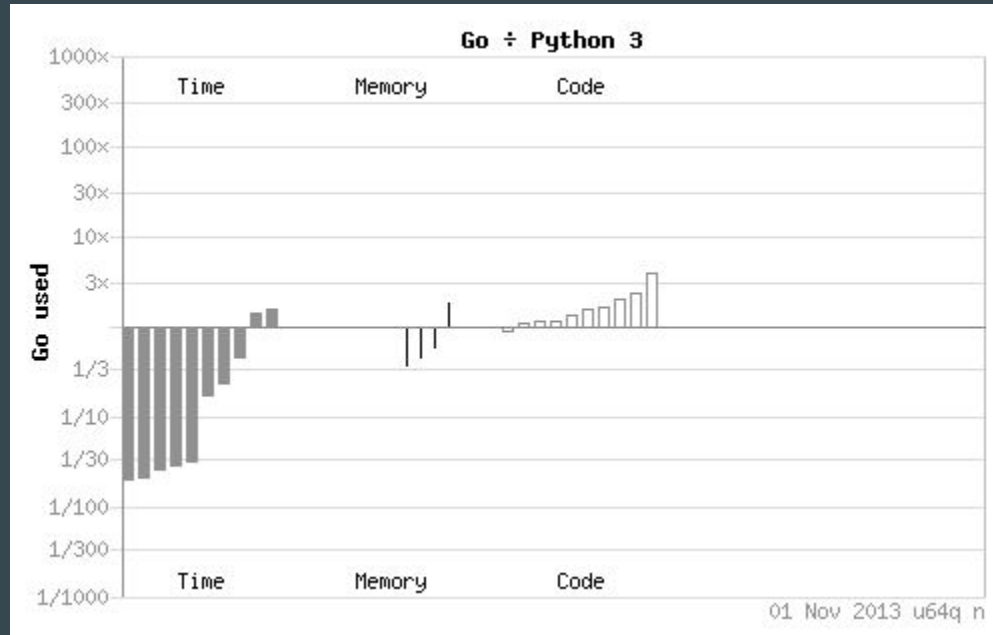
I got asked a lot: “why didn’t you use language X”?

Why not Python?

Seems logical coming from Ruby, but:

- Not compiled (more error prone)
- Indentation based code (more error prone)
- Not as fast as Go

Go vs Python Benchmark



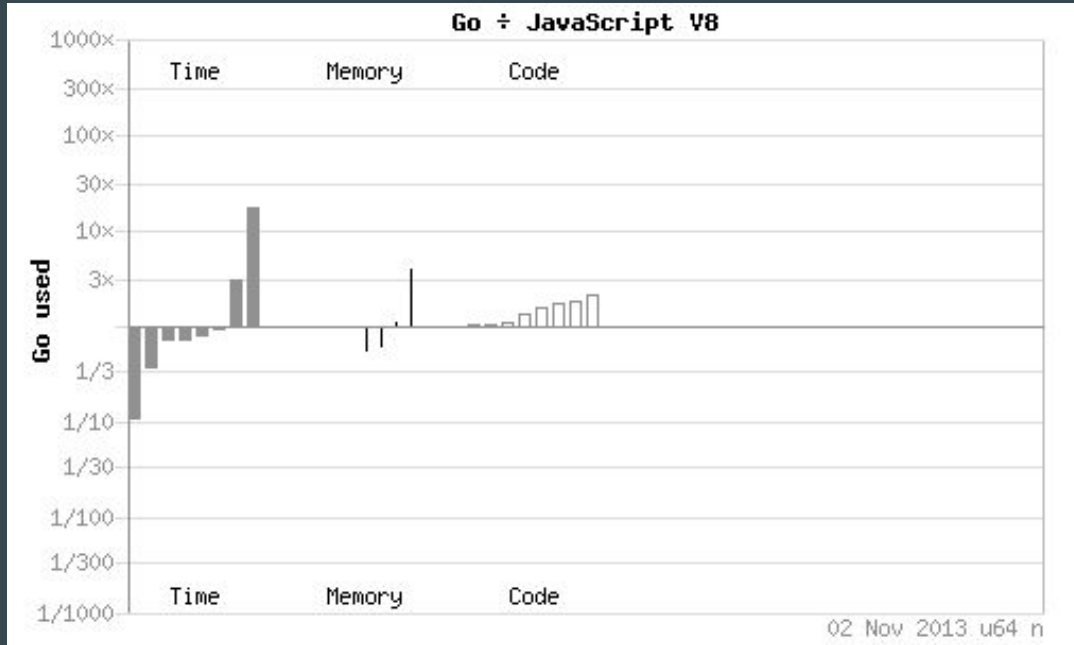
Why not Node?

JavaScript

When people tell me to use Node



JavaScript V8 is actually very fast



But it's still JavaScript.

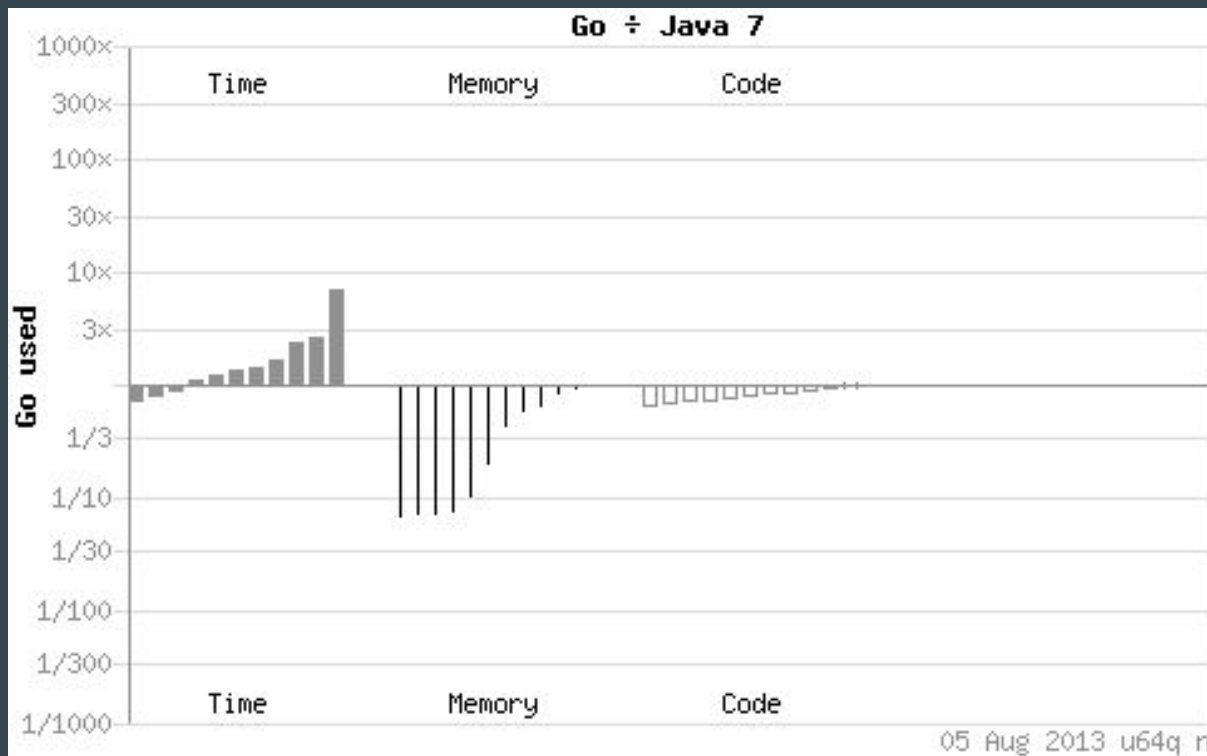
Why not Java (or a derivative)?

After many, many years of using Java, I didn't want to go back to the JVM.

Go was more interesting and exciting.

Even though Java is still faster.

Go vs Java Benchmark



Why not Ruby?

Some people asked why we didn't just try to optimize Ruby.

I'm sure we could have done a lot better had we not used Rails.

But even so, there's no comparison:

It Was a Risky Decision

- New technology, not proven
- There wasn't a big community
- There wasn't a lot of open source projects
- There weren't any success stories of production usage
- We weren't sure if we could hire top talent
- We were one of the first companies to publicly say we were using it
- We were the first company to post a Go job
- It wasn't even at a 1.0 release

When we told our investors we wanted to rewrite in Go



Replaced API with Go

Exact same API.

Exact same functionality.

**We went from 30
servers to 2**

30 Servers to 2

The 2nd one was just for redundancy.

We were barely utilizing the machines (barely registered CPU).

We never had a colossal CF again.

When you reduce your server count by 10x



**Part 3:
4 Years Later**

Performance

Performance has been stellar.

We still only run a few servers for each of our API clusters... after 4 years of growth!

Go has never been our bottleneck, it's always something else (ie: database).

Memory

No virtual machine - starts fast and small.

IronMQ starts up in 6.5MB of resident memory including configuration, making connections, etc.

Four years in, we've never had a memory leak or problems related to memory.

Reliability

Hard to quantify, but...

Our Go applications are very robust.

Rarely have a failure/crash that wasn't related to some external problem.

Code tends to be cleaner and higher quality.

Strict compiler.

Do a lot with a small amount of code.

Deployment

Go compiles into a single, static binary file.

Deployment is simply putting that file on a server and starting it up.

No dependencies required.

No runtime required.

Binary is small. (IronMQ is ~6MB)

Rolling Back

Since it's a single binary, you just stop the process and start the previous binary.

No need to worry about dependencies changing.

Language and Tooling

Keeps getting better and better.

Better garbage collection.

Better tools (for debugging, etc).

More open source libraries.

Talent

When we first started, there were very few people that knew the language.

We didn't know if we'd be able to hire anybody!

But people really want to work with Go.

The caliber of talent that want to work for Iron.io is amazing.

When I think about Go



Part 4: The Growth of Go

6 Years Old

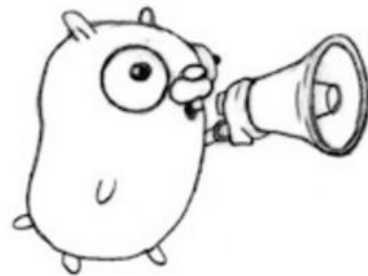
6 years and 6 days ago, the Go project was released. - Nov. 10, 2009

3.5 years ago, Go 1.0 was released. - March 28, 2012

The Go Blog

Go version 1 is released

28 March 2012



Today marks a major milestone

Trends

Compare Search terms ▾

Go

Programming La...

Java

Computer progra...

Ruby

Programming La...

Node.js

Software Develop..

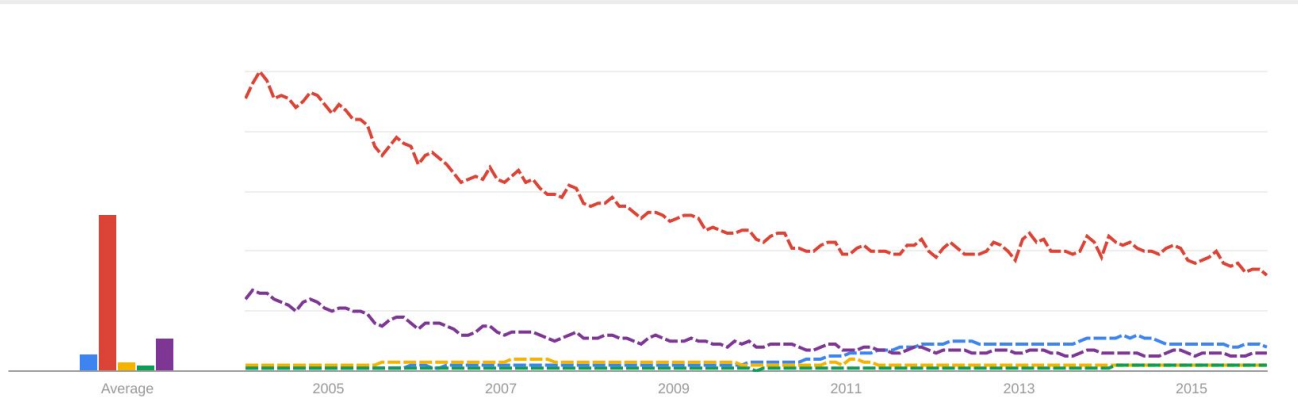
C++

Programming La...

Beta: Measuring search interest in *topics* is a beta feature which quickly provides accurate measurements of overall search interest. To measure search interest for a specific *query*, select the "search term" option. [?](#)

Interest over time [?](#)

News headlines [?](#) Forecast [?](#)



Trends

Compare Search terms ▾

Go

Programming Lan...

Ruby

Programming Lan...

Node.js

Software Developer

.NET Fram...

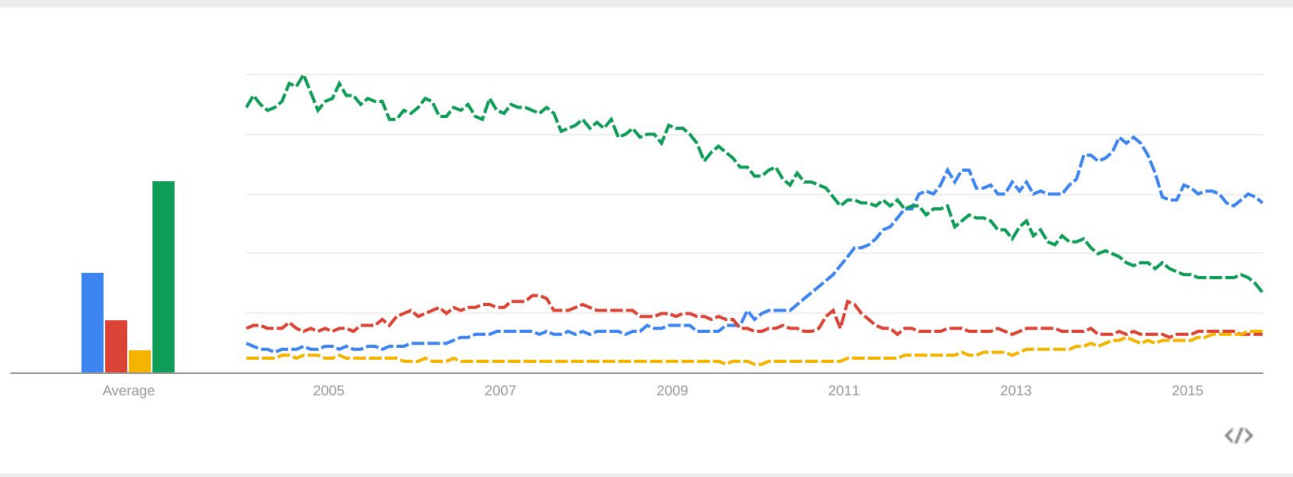
Software Framewo..

+ Add term

Beta: Measuring search interest in *topics* is a beta feature which quickly provides accurate measurements of overall search interest. To measure search interest for a specific *query*, select the "search term" option. [?](#)

Interest over time [?](#)

News headlines [?](#) Forecast [?](#)



Community



#GOSF

[Change photo](#)

San Francisco, CA

Founded Oct 24, 2011

[About us...](#)

[+ Invite friends](#)

Gophers

2,581

Gopherfest SV – w/Rob Pike, Travis Reeder, Blake Mizerany, Parse, and Dropbox

[Edit](#) [Cancel](#) [Feature](#) [Copy](#) [Ticket](#) [Export](#)
[Tell a friend](#) [Share](#)

Wednesday, November 18, 2015
6:30 PM

Parse
1 Hacker Way, Building 15, Menlo Park, CA ([edit map](#))

Your RSVP: Yes

[CHANGE](#)

[INVITE A FRIEND](#)

Tools

445 going



Travis Reeder

ORGANIZER

EVENT HOST

Co-founder and CTO of Iron.io

[Edit your intro](#)



Francesc Campoy

[View profile](#)

Production Usage

4 years ago, nobody was using it in production (except maybe Google... and us).

Today, almost every startup I talk to is using it in some way or another.

And a lot of the big guys are using it now too.

Who's Using Go Now?



Thousands of engineers writing Go code.

Millions of lines of Go source in the Google codebase.

High-profile and/or open source projects:

- Kubernetes
- Vitess.io
- Google Data Saver
- The unannounced Vanadium project - <https://v.io/> - source code here: <https://github.com/vanadium>

Who's Using Go Now?



Contributing to compilation and runtime for the Go language.

Looking to expose more Intel platform features through Go libraries.

Prototyping cloud ecosystem projects using Go.

Hosting Golang training workshops for Women In Technology.

Who's Using Go Now?



Replaced entire HTTP serving infrastructure with a Go stack.

> 100B requests per day.

~ 1.15M queries per second.

Who's Using Go Now?



Docker and all the Docker tools are written entirely in Go.

Docker announced at GoSF:



Who's Using Go Now?



#2 language (#1 being Ruby).

Services using Go:

- Dashboard metrics
- System metrics
- Logging system
- 'git push heroku master': The ssh server behind this is written in Go.
- Postgres server monitoring

“We're an infrastructure company and Go lends itself well to writing infrastructure code.” -- Edward Mueller

Who's Using Go Now?



Jamie Turner @jamwt · Aug 7
All the #golang excitement on hacker news makes me realize I don't think people realize how deep @Dropbox is in golang.

118 110

Jamie Turner @jamwt Follow

Our entire infrastructure service layer has been rewritten in golang, 10s of thousands of servers running millions of hits per second.

RETWEETS 59 FAVORITES 46

11:56 AM - 7 Aug 2015

Reply to @jamwt

Jamie Turner @jamwt · Aug 7
We have all our block storage on a home grown multi-datacenter distributed storage system, written in go.

32 33

Jamie Turner @jamwt · Aug 7
Exabytes of data, moving around Tbps flows 24/7.

Who's Using Go Now?

Server-side is 100% Go.

Including real-time chat service.

> 50M active users.

13 billion minutes/month.



Conclusion

We took a risk and it paid off.

We love the language, and it's loved us back.

Usage is growing super fast.

A lot of companies are already using it for critical parts of their production systems.

Conclusion

2 years ago I wrote: “Is Go the next gen language we’ve been waiting for? It’s a bit too early to say, but it’s certainly off to a good start.”

Today: Yes it is. Go IS the language for the cloud.

If you’re writing API’s or infrastructure, it should be at the top of your list.

Is Go the new Java? It’s a bit too early to say, but it’s certainly off to a good start.

Thanks!



Travis Reeder
@treeder
travis@iron.io

