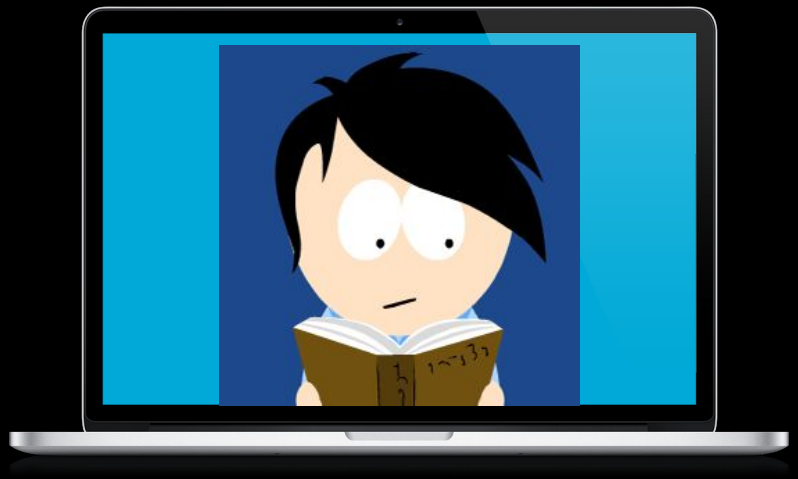
Refactoring to modules

Why, How and Everything Else

JFrog

Who am I?

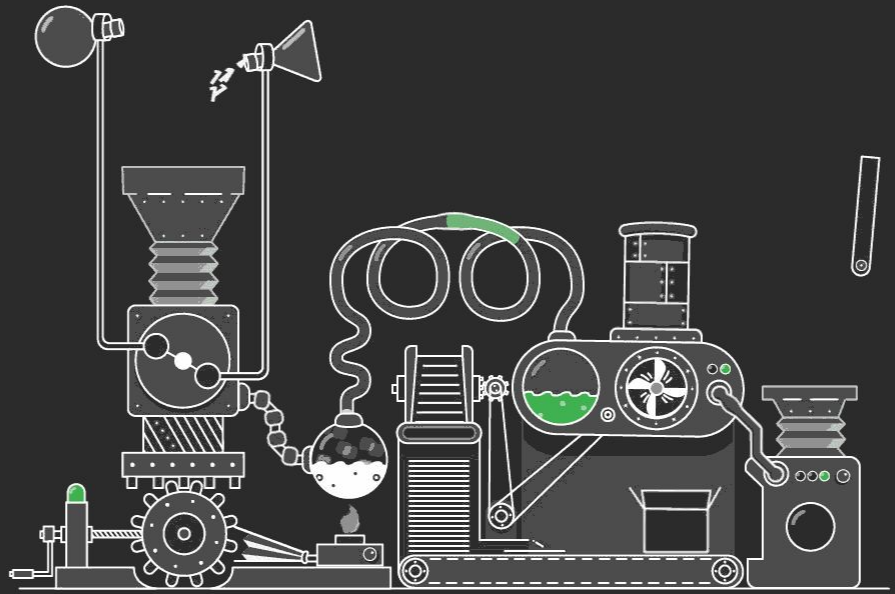


Elad Hirsch , DevOps Consultant

- DevOps Consultant
- Full-stack software engineer
- Tackling everything from **DevOps** and **Backend** challenges to the latest **Frontend** frameworks

FROGS - THE LIQUID SOFTWARE COMPANY

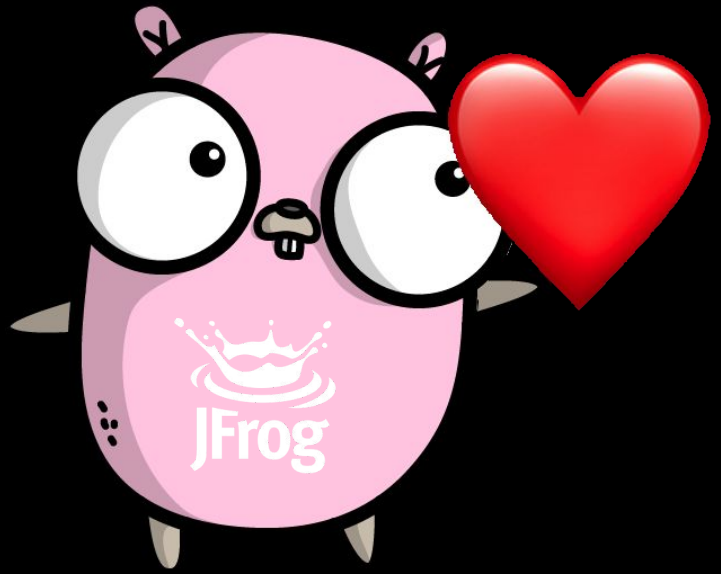
JFrog Artifactory is getting ready to work...



JFrog

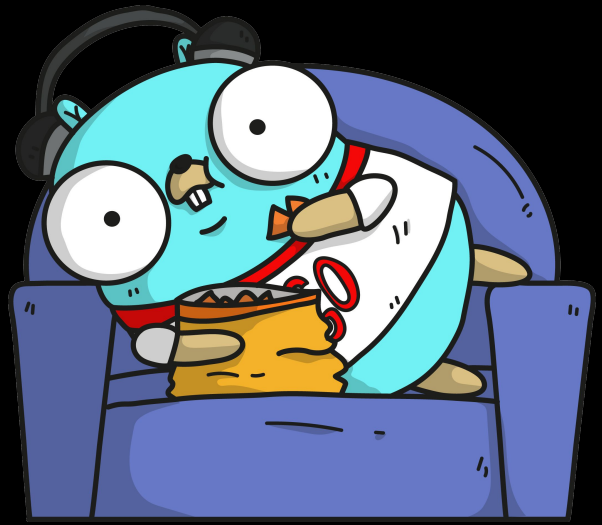
We absolutely love Go and we use it a lot

GoCenter
Xray
JFrog CLI
Metadata server
Replicator
E+ Platform Installer



Let’s travel back in time…

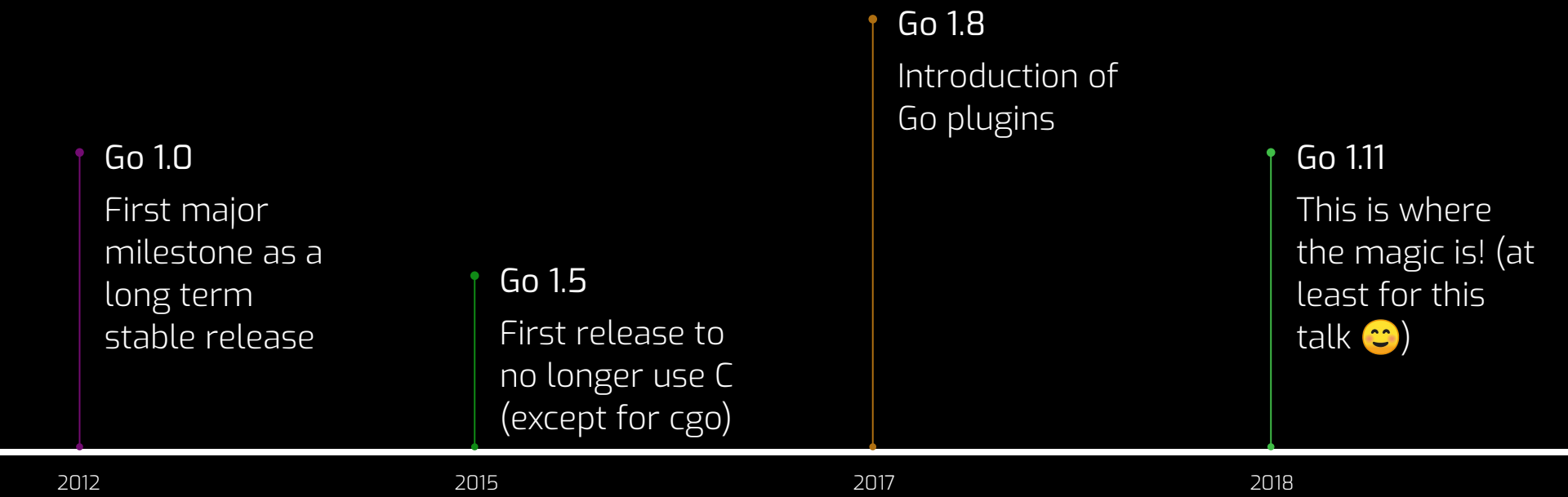
Let’s turn to the audience for a poll…



When did you start with Go?

1.0	1.2	1.5	1.8	1.11
2012	2013	2015	2017	2018

A quick history of go



So what is that magic?

Dependency Management...

"*Tis impossible to be sure of anything but Death and Taxes*"

- Christopher Bullock

Let’s start with the why…

Why do we have this problem?

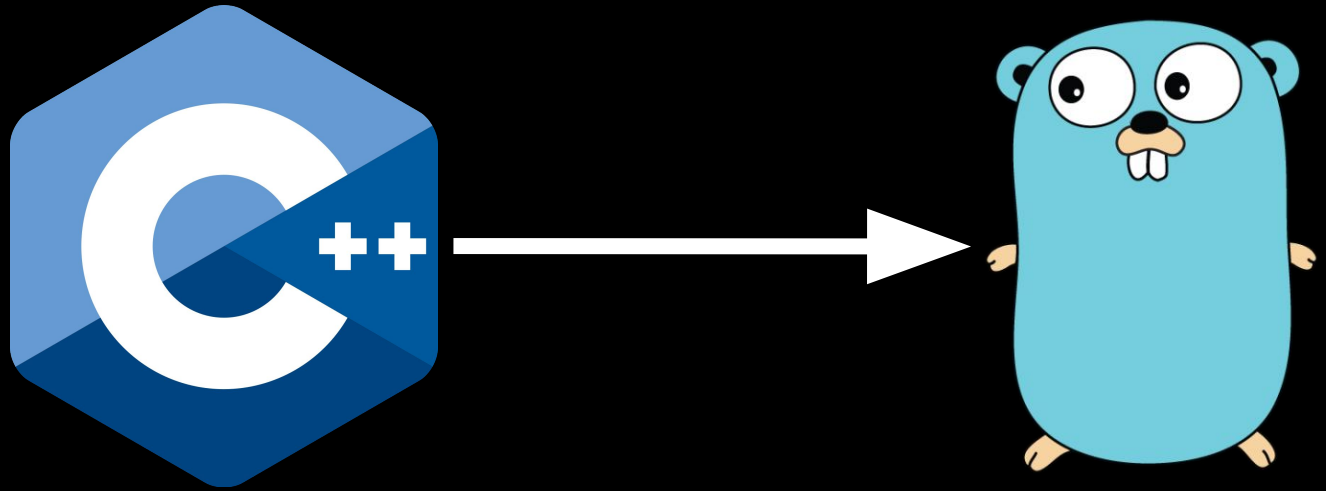
History

Design began in late 2007.

Key players:

- Robert Griesemer, Rob Pike, Ken Thompson
- Later: Ian Lance Taylor, Russ Cox

GO origins



Dependency management @ Google

- One Huge Makefile
- Refactor into multi Makefile per module
- Pain of dependency management is huge this days

Dependency Hell term is born

Let’s do something very simple

One easy solution…?

Dependencies are sources!

Remote imports are in VCS

Dump everything into a single folder

Compile everything together

Profit!!

Wait ! But… how do I know…

- Which dependencies I use?
- Which dependencies you used?
- Which dependencies I should use?
- Which code I'm editing right now?
- Is there any dependency duplications?
- What is going on?!

The official response - Let's duplicate dependencies

"Check your dependencies to your own VCS."

Andrew Gerrand

Vendoring – the worst kind of forking

"Copy all of the files at some version from one version control repository and paste them into a different version control repository"

But what is wrong with vendoring?

- History, branch, and tag information is lost (metadata is lost)
- Pulling updates is impossible
- It invites modification, divergence, and bad fork
- It wastes space
- Good luck finding which version of the code you using

Build your own dependency manager…

"It's not the role of the tooling provided by the language to dictate how you manage your code (...)"

Andrew Gerrand

The community steps in


AND THE NEXT THING YOU KNOW...
LIGHTYEAR
THERE ARE 19 DEPENDENCY MANAGERS

So you want to write a package manager

You woke up this morning, rolled out of bed, and thought, "Y'know what? I don't have enough misery and suffering in my life. I know what to do—I'll write a language package manager!"

...

Package management is awful, you should quit right now

Package management is a nasty domain. Really nasty. On the surface, it *seems* like a purely technical problem, amenable to purely technical solutions. And so, quite reasonably, people approach it that way. Over time, these folks move inexorably towards the conclusion that:

1. software is terrible
2. people are terrible
3. there are too many different scenarios
4. nothing will really work for sure
5. it's provable that nothing will really work for sure
6. our lives are meaningless perturbations in a swirling vortex of chaos and entropy

★ Top highlight

Go dep – Proper dependency management?

- Working in project directories
- Local cache for dependencies
- Version declarations
- Conflict resolution

Go dep status

- Started as an experiment
- Lesson learned from - java / npm ...
- Main challenge - how to solve conflict resolution
  - Dep disallow multiple major versions
  - GO Tech lead *Russ Cox* took another approach
  - Go -> Simple solution -> **SMV**

Tweet

Replying to @_rsc @mattfarina and 2 others

Although it was a successful experiment, Dep is not the right approach for the next decade of Go development. It has many very serious problems. A few:

12:14 AM · Jul 27, 2018 · Twitter Web Client

1 Retweet **9** Likes

Russ Cox @_rsc · Jul 27, 2018

Replying to @_rsc @mattfarina and 2 others

- Dep does not support using multiple major versions of a program in a single build. This alone is a complete showstopper. Go is meant for large-scale work, and in a large-scale program different parts will inevitably need different versions of some isolated dependency.

1 1 18

And according to the dep project page:

> *dep was the "official experiment." The Go toolchain, as of 1.11, has (experimentally) adopted an approach that sharply diverges from dep. As a result, we are continuing development of dep, but gearing work primarily towards the development of an alternative prototype for versioning behavior in the toolchain.*

For more information about the new Go build-in management, please refer to the official GitHub Wiki — Go 1.11 Modules.

Thanks John Arundel @bitfield and Erhan Yakut @yakuter for revealing the problem. 🙇

—

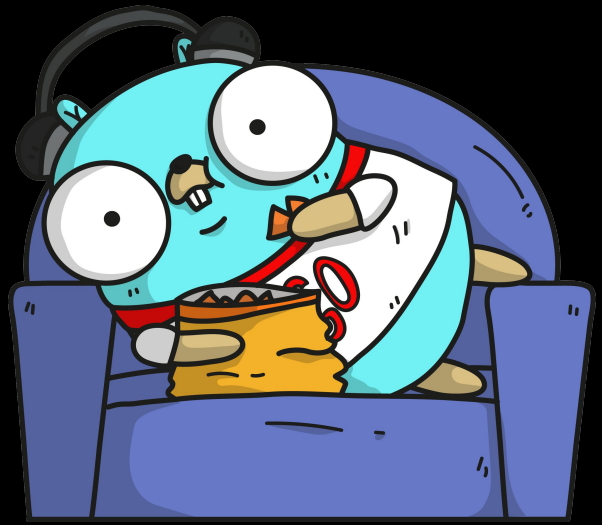
Update @ 2018–02–03: Sam Boyer from the godep team has clarified some incorrect information in this article. I apologize to Sam Boyer and the readers for any inconvenience. 😢


icanhasGIF.com
Now what?

Let’s actually go fix it!

GO Modules

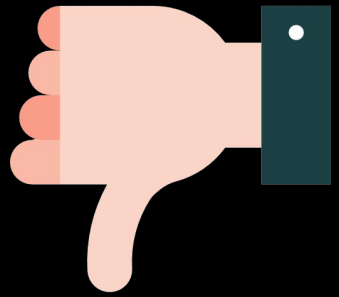
Let’s turn to the audience for another poll…



Who is using Go modules?

Yes

No



Let’s go over some definitions

Module

A module is a collection of related Go packages that are versioned together as a single unit.

Let’s go over some definitions

Sources

A module is a tree (directory) of Go source files with a `go.mod` file in the root directory.

Let’s go over some definitions

Version Control

Most often, a single version-control repository corresponds exactly to a single module

Using go modules

Work inside $GOPATH

Set the environment variable
`GO111MODULE` to `ON`

Work outside $GOPATH

you do not need to set
`GO111MODULE`

Creating a module

```
mkdir mymodule
cd mymodule
go mod init github.com/retgits/mymodule
```

Moving from DEP to modules

- Import dependencies from `Gopkg.lock`

```
go mod init <module path>
```

- Remove unnecessary imports and add indirect imports

```
go mod tidy
```

- Delete the vendor folder

```
rm –rf vendor/
```

- Delete Gopkg files

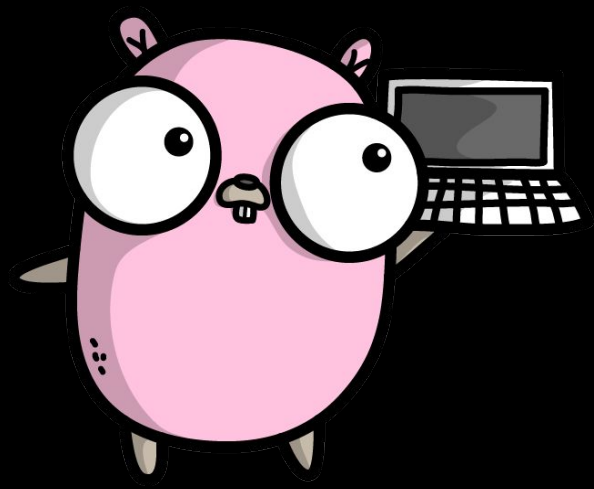
```
rm Gopkg.*
```

Upgrade from dep to go modules

git clone https://github.com/eladh/sample-app-go-dep.git

Versioning

```
module github.com/retgits/mymodule

go 1.12

require (
        github.com/naoina/go-stringutil v0.1.0
        github.com/some/dependency v1.2.3
        github.com/google/go-github/v25 v25.0.1
)
```



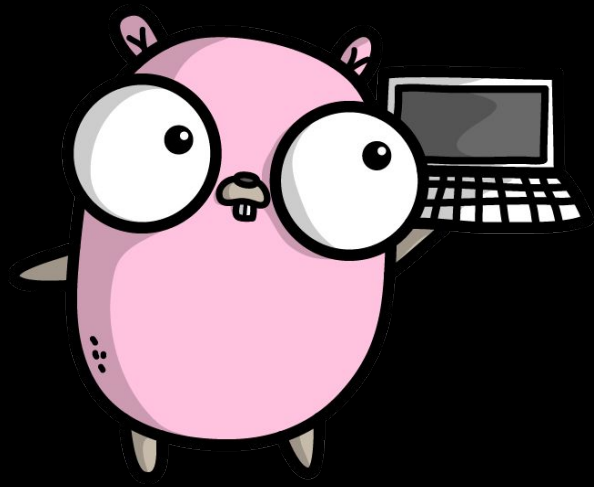
1 . 3 . 1

BREAKING . FEATURE . FIX

incompatible API changes	add backwards-compatible functionality	make backwards-compatible bug fix
breaking change	new **feature**	bug **fix**

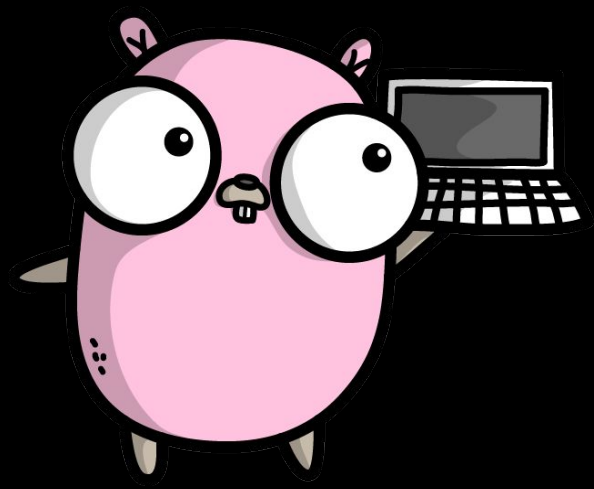
Replacing packages

```
module github.com/retgits/mymodule

go 1.12

require (
        github.com/naoina/go-stringutil v0.1.0
        github.com/some/dependency v1.2.3
        github.com/google/go-github/v25 v25.0.1
)

replace github.com/some/dependency github.com/retgits/dependency
```



It’s great when you got dependencies that still under local development

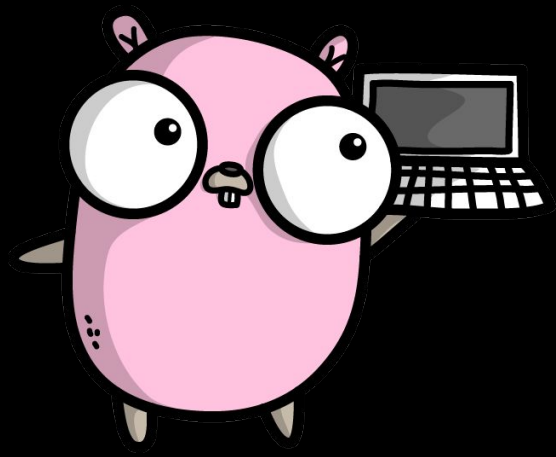
```
module github.com/retgits/mymodule

go 1.12

require (
        github.com/naoina/go-stringutil v0.1.0
        github.com/some/dependency v1.2.3
        github.com/google/go-github/v25 v25.0.1
)

replace github.com/some/dependency ../../Downloads/dependency
```



But I kinda like the vendor folder? It's provide us dependencies beforehand

- You can create the vendor folder
  ```
  go mod vendor
  ```
- You can use the vendor folder during builds
  ```
  go build -mod=vendor
  ```

This does mean you need to have all dependencies listed in your go.mod file



Go get my a new super duper module

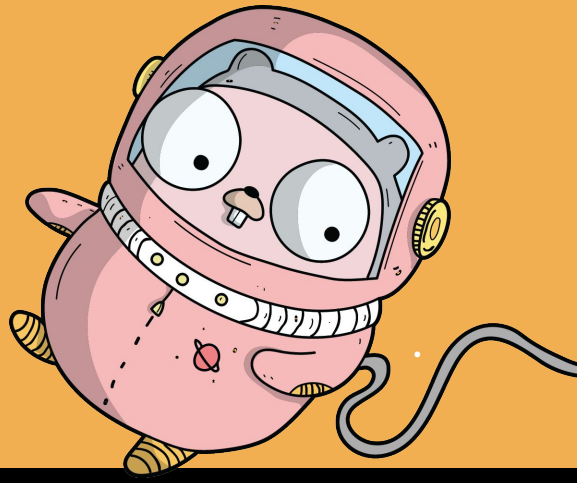
Before modules

```
go get https://github.com/sirupsen/logrus
git clone https://github.com/sirupsen/logrus
<build module locally>
```

With modules

```
go get github.com/sirupsen/logrus
GET github.com/sirupsen/logrus/@v/list
GET github.com/sirupsen/logrus/@v/v1.0.0.mod
GET github.com/sirupsen/logrus/@v/v1.0.0.zip
```

Houston… I think I lost my module?



Modules are immutable

The **<module>@v<version>** construct should be immutable

That means that

`github.com/retgits/checkiday/@v/v1.0.0`

Should forever be the same...

But are they really?

"Friends don't let friends do git push -f"

- Aaron Schlesinger

Using the *GOPROXY* variable

```
export GOPROXY=https://myawesomeproxy.com
go get github.com/retgits/checkiday
```

A Go module proxy is any web server that can respond to GET requests for URLs of a specified form.

The requests have no query parameters, so even a site serving from a fixed file system (including a file:/// URL) can be a module proxy.

Keeping modules

Local cache (**`$GOPATH/pkg/mod`**)

Immediate access, not shared, can be wiped...

Organizational cache (private proxy)

Fast access, requires infra, shared across devs

JFrog ARTIFACTORY

THE Athens PROJECT

Public cache (public proxy)

Highly available, CDN, no infra, free

JFrog GoCenter

And also faster builds…

Let’s check public proxy performance

```
Setting GoProxy to
  0.54s user 0.72s system 38% cpu 3.320 total
  0.54s user 0.73s system 38% cpu 3.282 total
Setting GoProxy to https://gocenter.io
  0.28s user 0.22s system 9% cpu 5.613 total
  0.28s user 0.22s system 21% cpu 2.388 total
```

JFrog GoCenter The Central Go Modules Repository

ReadMe

Add

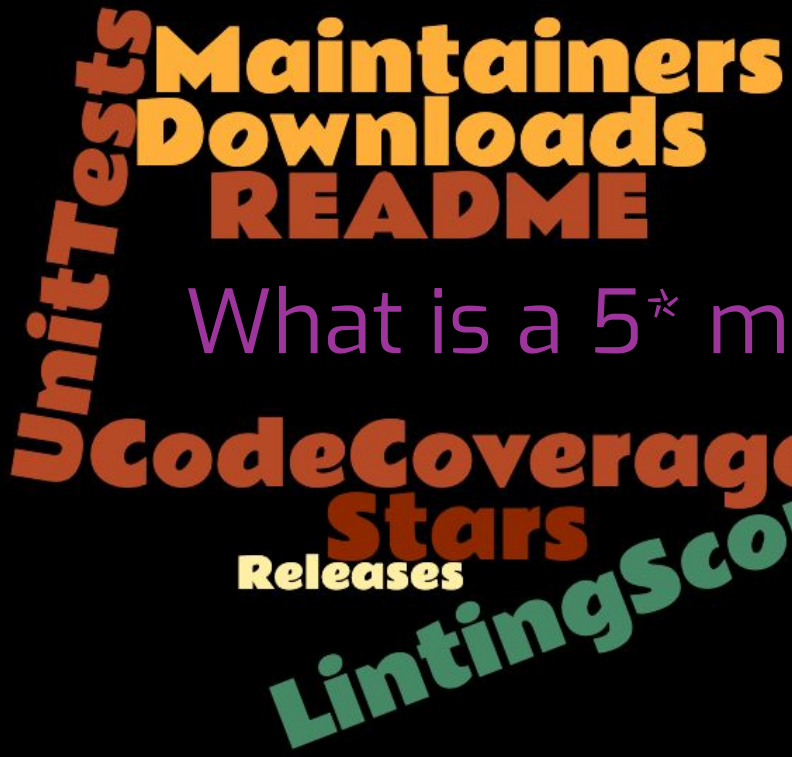
106,797

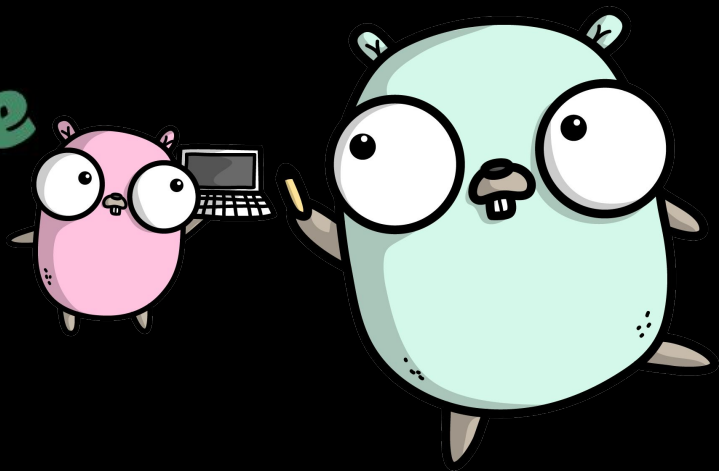
Versions indexed

Feedback

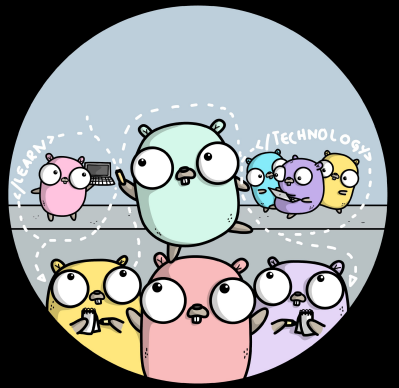
Contact Us Privacy Policy Terms Of Service Status

How can we ensure module quality ?





Final thoughts



- If you haven't, start with Go modules:
  `GO111MODULE=on` if you must stay on **`$GOPATH`**
- Start working outside of your **`$GOPATH`**
- Use a public proxy server for immutable Go modules (l**ike GoCenter.io**)
- Think about running your own proxy server or repository manager such as **Artifactory** or **Athens**

Go modules are in beta stage ?

*Modules are supported from Go version **1.11***
*but it will be finalized in Go version **1.13***

Labels | Milestones | New issue

Go1.13

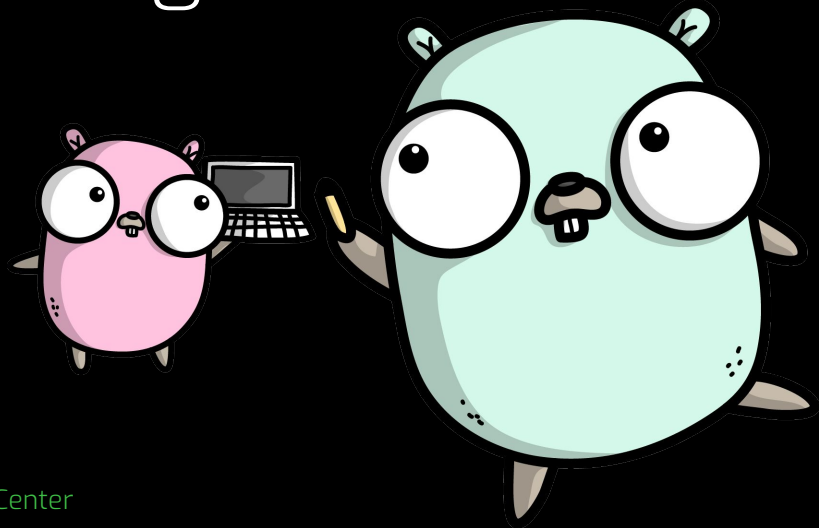
Due by August 01, 2019 **97%** complete

19 Open ✓ 631 Closed

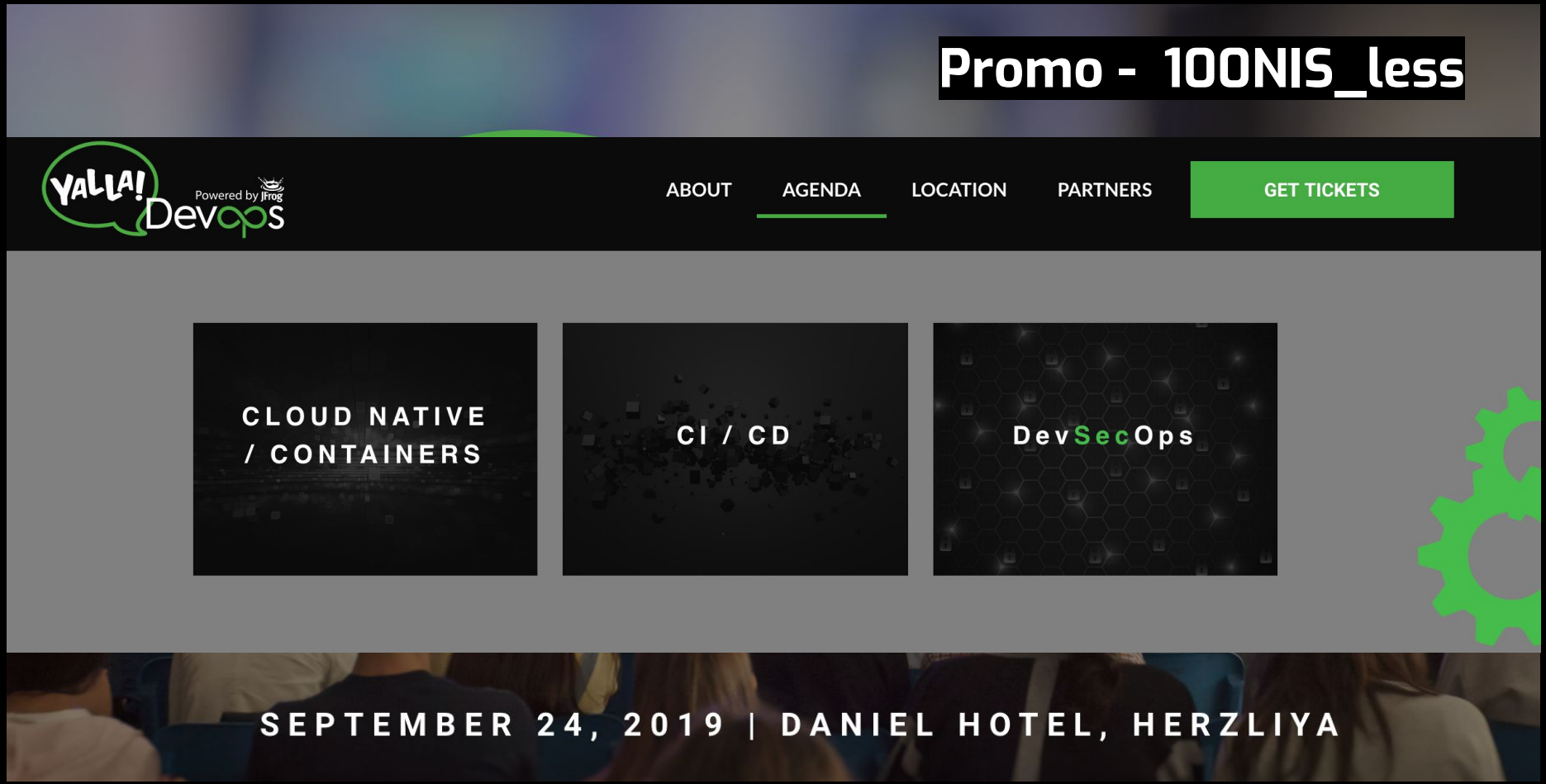
- cmd/go: 'go list -e -json /absolute/path' doesn't show correct import path — 4
  GoCommand NeedsInvestigation
  #33157 opened 9 days ago by jayconrod
- net/http: Header.Clone behavior NeedsDecision release-blocker
  #33141 opened 10 days ago by dsnet
- archive/zip: reject certain invalid archives NeedsInvestigation — 15
  #33026 opened 16 days ago by MichaelTJones
- proxy.golang.org: '400 bad request' for pseudo-versions that refer to commits that also have canonical version tags NeedsFix WaitingForInfo modules — 13
  #32879 opened 25 days ago by it-philpennock
- doc: the Go 1.13 release notes don't mention the new "%w" verb in fmt — 4
  Documentation NeedsFix release-blocker
  #32914 opened 24 days ago by ainar-g
- runtime: linux/arm64 crash in runtime.sigtrampgo NeedsInvestigation release-blocker — 21
  #32912 opened 24 days ago by jing-rui

But before I go…

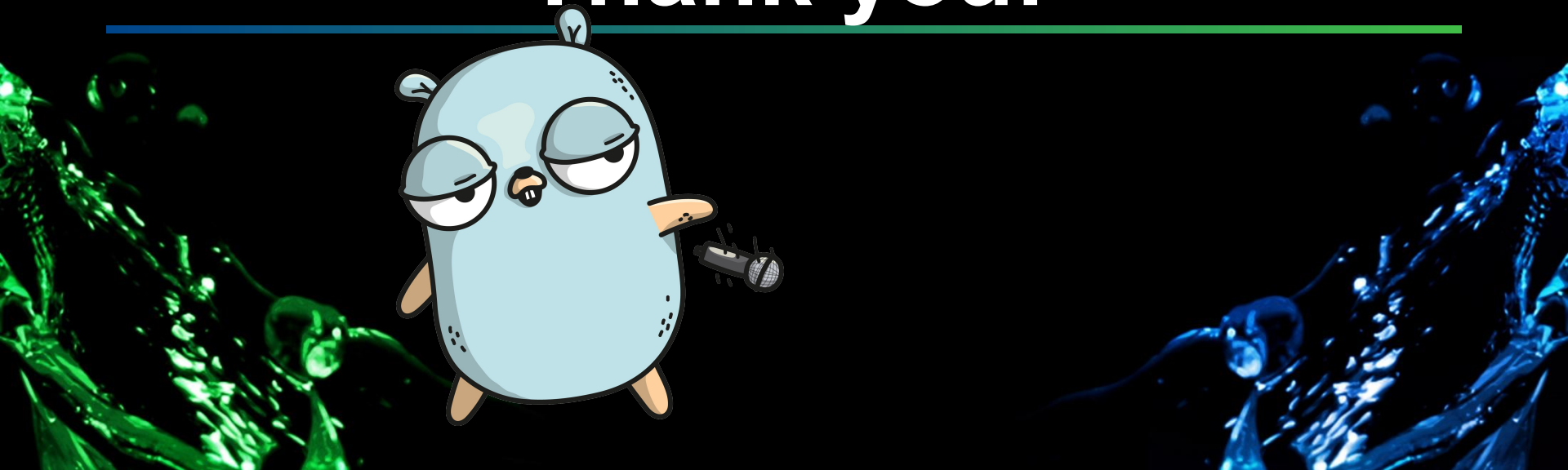
There's one more thing…



Promo - 100NIS_less



Thank you!



JFrog CLI - GO support

```
// Configure Artifactory:
jfrog rt c

//Configure the project's repositories::
jfrog rt go-config

//Build the project with go and resolve the project dependencies from
Artifactory.
jfrog rt go build --build-name=my-build --build-number=1

//Publish the package we build to Artifactory.
jfrog rt gp go v1.0.0 --build-name=my-build --build-number=1

//Collect environment variables and add them to the build info.
jfrog rt bce my-build 1

//Publish the build info to Artifactory.
jfrog rt bp my-build 1
```