O'REILLY®

Practical Go Modules

About the trainer

bmuschko

bmuschko

bmuschko.com

AUTOMATED ASCENT

automatedascent.com

DISCUSSION

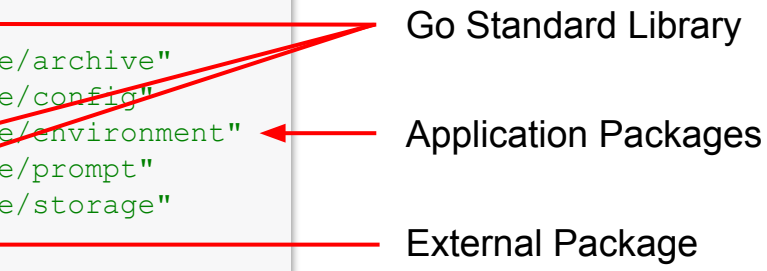
What’s your main learning objective?

Why Modules?

Understanding Go’s Native Package Manager and the Problems it Solves

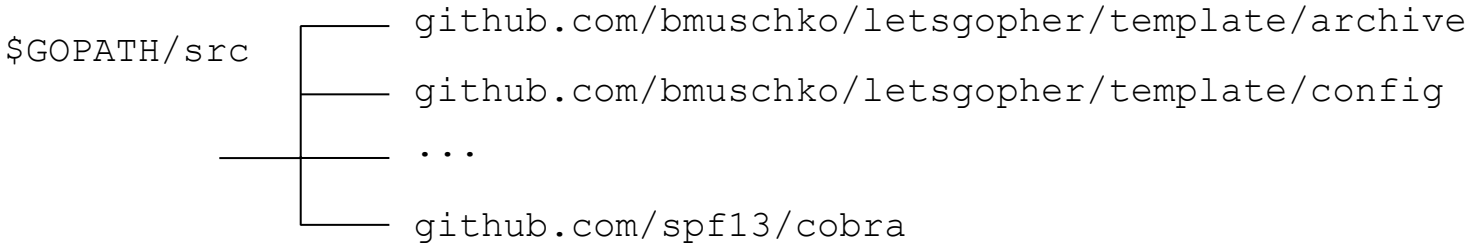
Importing Packages

```
package cmd

import (
      "fmt"
      "github.com/bmuschko/letsgopher/template/archive"
      "github.com/bmuschko/letsgopher/template/config"
      "github.com/bmuschko/letsgopher/template/environment"
      "github.com/bmuschko/letsgopher/template/prompt"
      "github.com/bmuschko/letsgopher/template/storage"
      "github.com/spf13/cobra"
      "io"
      "strings"
)
```



Resolved from GOPATH

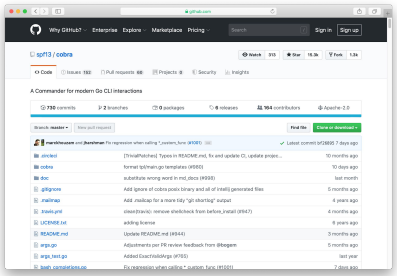
Environment variable required to set up Go



Retrieving External Packages

Downloads the latest commit from `master` branch

`go get`



https://github.com/spf13/cobra

download

`$GOPATH/src`

`$GOPATH/pkg`

`go install`

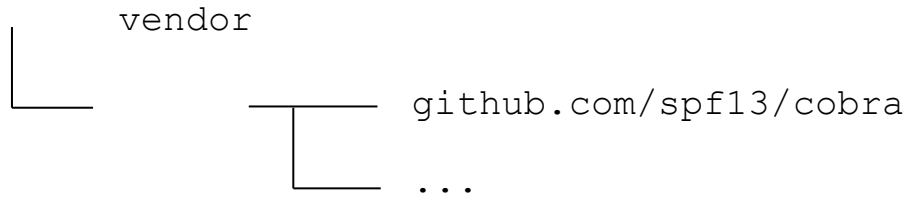
DISCUSSION

What problems can arise?

Vendoring to the Rescue?

Special directory in project source tree checked into SCM

```
$GOPATH/src/github.com/bmuschko/letsgopher
```



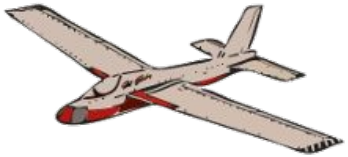
Automatically uses packages from vendor directory

DISCUSSION

Can you identify potential issues?

Open Source Package Managers

Community solutions for defining & resolving dependencies



Glide

dep

gopkg.in

Stable APIs for the Go language

and many more...

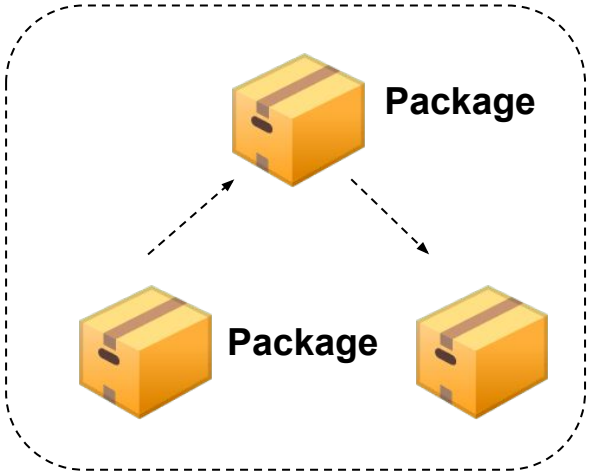
Goals of Modules

Reliable and repeatable builds

- Project source code can be checked out independent of `GOPATH`
- Retrieval and usage of versioned, external dependencies
- Full integration with Go tool chain

What's a Module?

A collection of packages with potential dependencies



Module

manages

`go.mod`

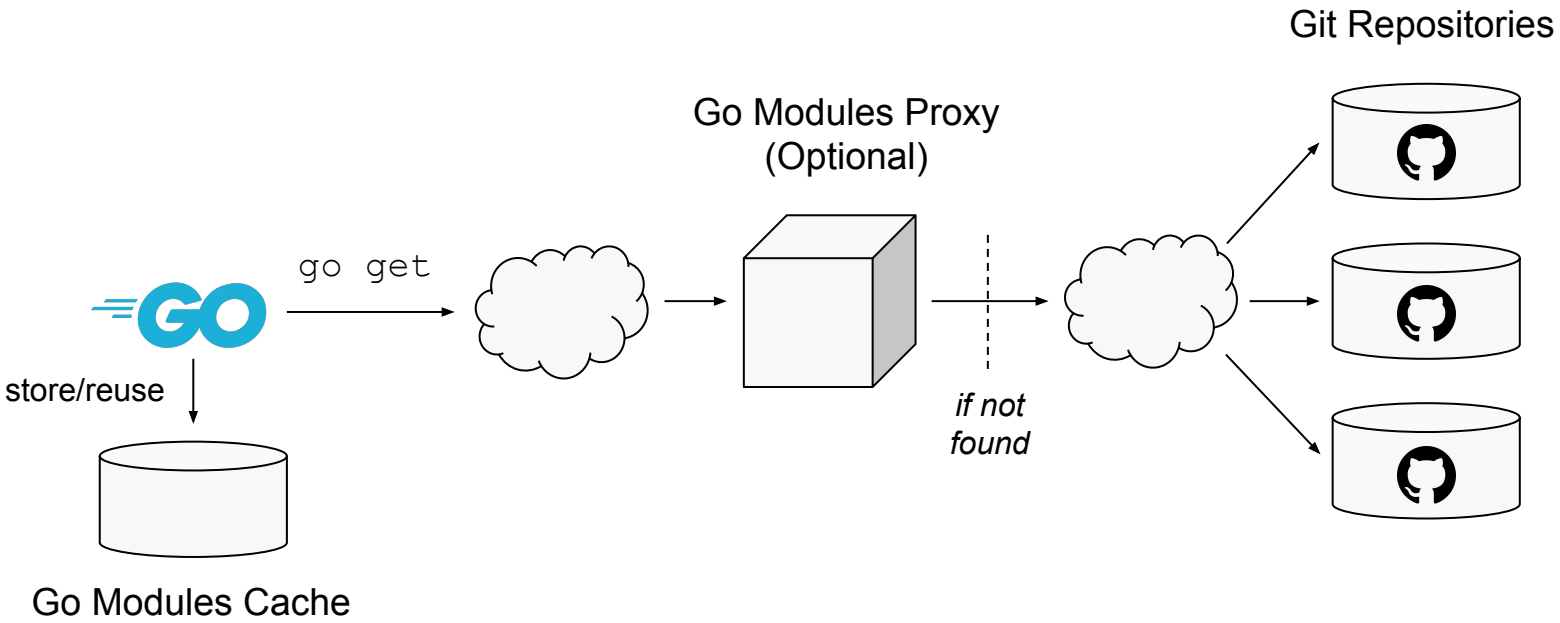
Characteristics of Modules

Focus on performance and integrity

- Dependencies are identified by a semantic version
- Go tries to resolve dependencies from a central proxy first
- Downloaded dependencies are cached locally and reused
- Checksums are used to ensure integrity of a dependency

Dependency Resolution Workflow

Proxy and cache help with reproducibility and performance



Anatomy of a go.mod file

Definition to be checked into SCM and evolving over time

```
module github.com/bmuschko/letsgopher

require (
      github.com/mattn/go-runewidth  v0.0.4 // indirect
      github.com/mitchellh/go-homedir  v1.0.0
      github.com/spf13/afero  v1.2.2 // indirect
      github.com/spf13/cobra  v0.0.3
      github.com/spf13/pflag  v1.0.3 // indirect
)

go 1.20
```

Root package of application

Definition of dependencies

Expected Go version to be used for project

Dependency Checksums

Cryptographic hashes are stored in `go.sum`

```
github.com/mitchellh/go-homedir v1.1.0/go.mod h1:SfyaCUpYCn1Vlf4IUYiD9fPX4A5wJrkLzIz1N1q0pr0=
github.com/mitchellh/mapstructure v1.1.2/go.mod h1:FVVH3fgwuzCH5S8UJGiWEs2h04kUh9fWfEaFds41c1Y=
github.com/pelletier/go-toml v1.2.0/go.mod h1:5z9KED0ma1S8pY6P1sdut58dfprrGBbd/94hg7ilaic=
github.com/pmezard/go-difflib v1.0.0/go.mod h1:iKH77koFhYxTK1pcRnkKkqfTogsbg7gZNVY4sRDYZ/4=
github.com/russross/blackfriday v1.5.2/go.mod h1:JO/DiYxRf+HjHt06OyowR9PTA263kcR/rfWxYHBV53g=
github.com/spf13/afero v1.1.2/go.mod h1:j4pytiNVoe2o6bmDsKpLACNPDBIoEAkihy7loJ1B0CQ=
github.com/spf13/cast v1.3.0/go.mod h1:Qx5cxh0v+4UWYiBimWS+eyWzqEqokIECu5etghLkUJE=
github.com/spf13/cobra v0.0.5 h1:f0B+LkLX6DtmRH1isoNA9VTtNUK9K8xYd28JNNfOv/s=
github.com/spf13/cobra v0.0.5/go.mod h1:3K3wKZymM7VvHMDS9+Akkh4K60UwM26emMESw8tLCHU=
```

Module

Version

Hash based on file tree and download timestamp

Modules Quickstart

Exploring the Essential Functionality

Initializing a Module

Simple `go` command, run just once

Module Path


```
$ go mod init github.com/bmuschko/hello-world
go: creating new go.mod: module github.com/bmuschko/hello-world
```

Generates a new ✨ in current directory

`go.mod`

Introspecting the go.mod

File doesn’t contain any dependencies initially

```
module github.com/bmuschko/hello-world

go 1.20
```

The GOMODULE111 Env. Var

Enables Modules by default if `go.mod` can be found

```
GOMODULE111=auto
```

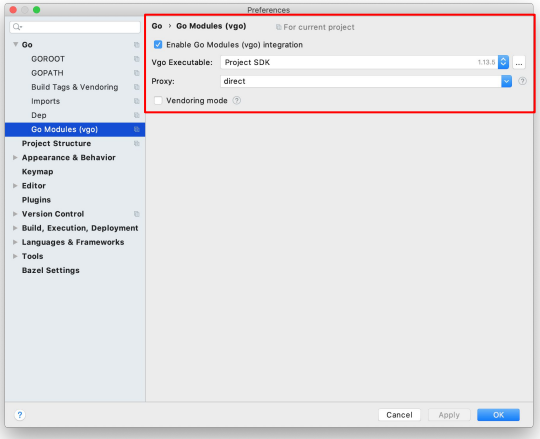
✓ **Modules Enabled**

✗ **Modules Disabled**

GO ***1.13*** **-** ***1.16***

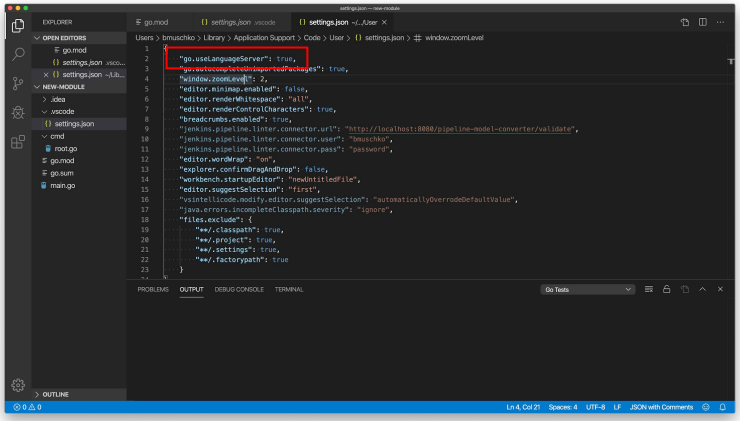
Enabling Modules in IDE

IDEs need to explicitly enable the option



GoLand

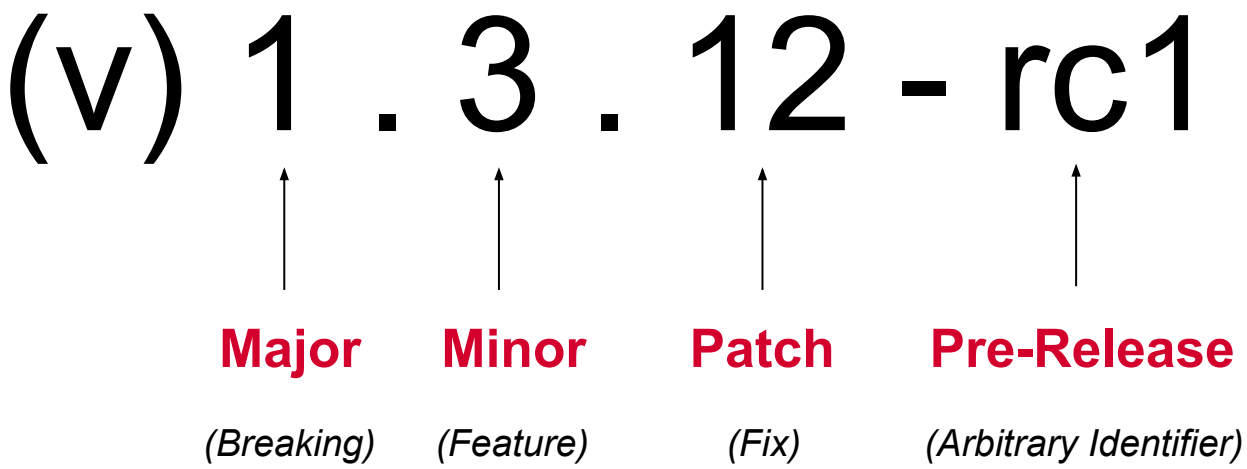
```
"go.useLanguageServer": true
```



VSCode

Semantic Versioning

Modules selects a package based on version

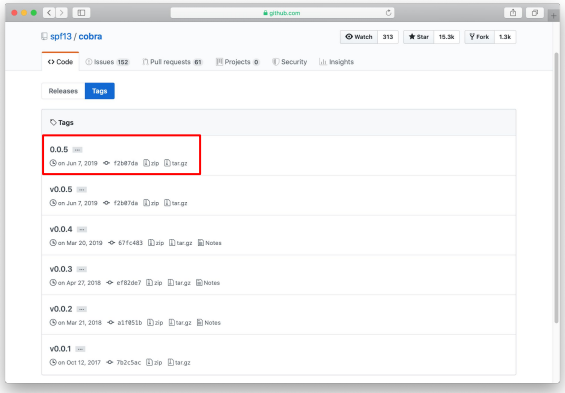


Adding a Dependency

Resolves the latest tag of dependency

```
$ go get github.com/spf13/cobra
go: finding github.com/spf13/cobra v0.0.5
go: downloading github.com/spf13/cobra v0.0.5
go: extracting github.com/spf13/cobra v0.0.5
```

https://github.com/spf13/cobra/tags



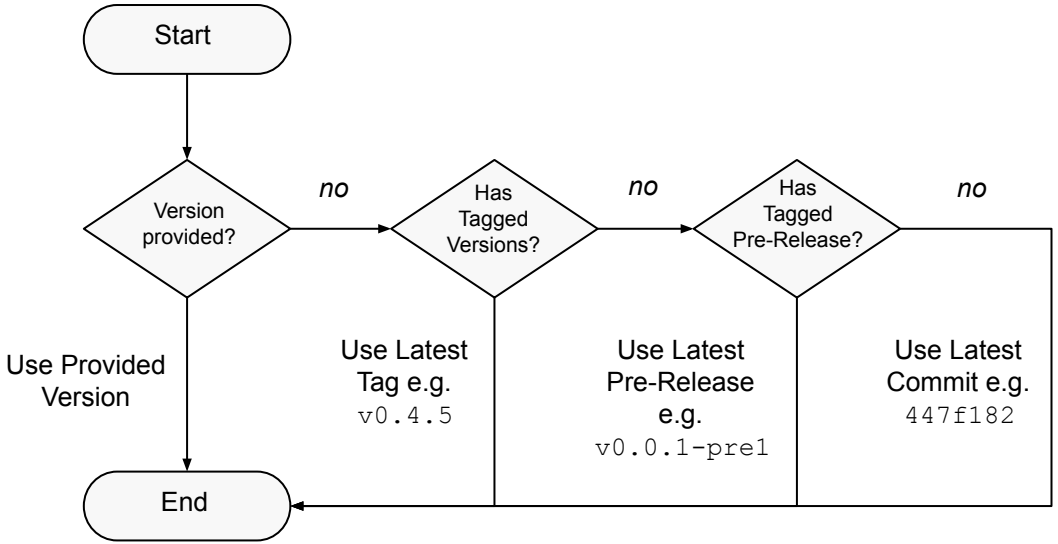
Resulting entry in go.mod

Adds dependency and any transitive dependencies

```
module github.com/bmuschko/hello-world

go 1.20

require github.com/spf13/cobra v0.0.5 // indirect
```

Automatic Version Selection

The `go get` command follows a version selection strategy



Listing Available Versions

“What versions have been released of this dependency?”

```
$ go list -m -versions github.com/spf13/cobra
github.com/spf13/cobra v0.0.1 v0.0.2 v0.0.3↵
v0.0.4 v0.0.5
```

Version JSON Representation

“Show me more information about the dependency!”

```
$ go list -m -json -versions github.com/spf13/cobra
{
      "Path": "github.com/spf13/cobra",
      "Version": "v0.0.5",
      "Versions": [
            "v0.0.1",
            "v0.0.2",
            "v0.0.3",
            "v0.0.4",
            "v0.0.5"
      ],
      "Time": "2019-06-07T14:48:23Z",
      "Dir": "/Users/bmuschko/go/pkg/mod/github.com/spf13/cobra@v0.0.5",
      "GoMod": "/Users/bmuschko/go/pkg/mod/cache/download/github.com/spf13/cobra/@v/v0.0.5.mod",
      "GoVersion": "1.12"
}
```

All available versions

Storage location in cache

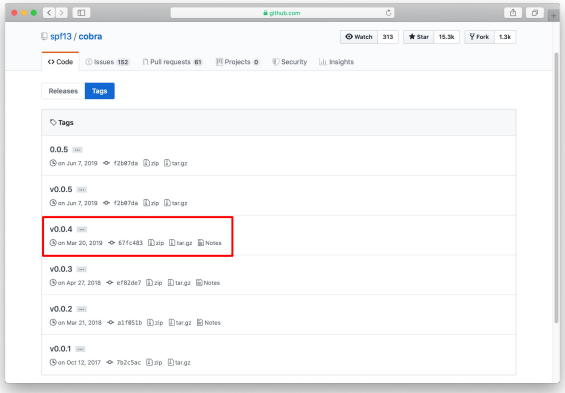
Go version of dependency

Selecting a Specific Version

*Good reasons why you might **not** want to pick latest*

```
$ go get github.com/spf13/cobra@v0.0.4
go: finding github.com/spf13/cobra v0.0.4
go: downloading github.com/spf13/cobra v0.0.4
go: extracting github.com/spf13/cobra v0.0.4
```

https://github.com/spf13/cobra/tags



Storage in Cache

Downloaded dependencies available in GOPATH

```
$ echo $GOPATH
/Users/bmuschko/go

$ tree $GOPATH/pkg/mod/github.com
github.com
└── spf13
    ├── cobra@v0.0.5
    │   ├── ...
    └── pflag@v1.0.3
        ├── ...

$ tree $GOPATH/pkg/mod/cache/download/github.com/spf13/cobra/@v
.
├── v0.0.5.mod
├── ...
```

Go Path env. var containing cache

Downloaded Go files of dependencies

Go module files of dependencies

EXERCISE

Initializing and Using
Modules for a Project

Q & A

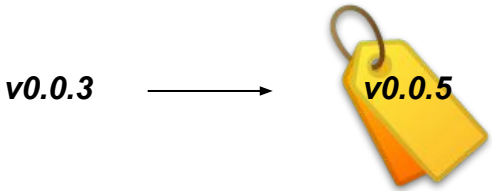
BREAK

Modules Day-to-Day Workflows

Evolving a Project with Modules

Upgrading Dependency Versions

“I want to use a different, non-breaking released version.”




```
$ go get
github.com/spf13/cobra@v0.0.5
```

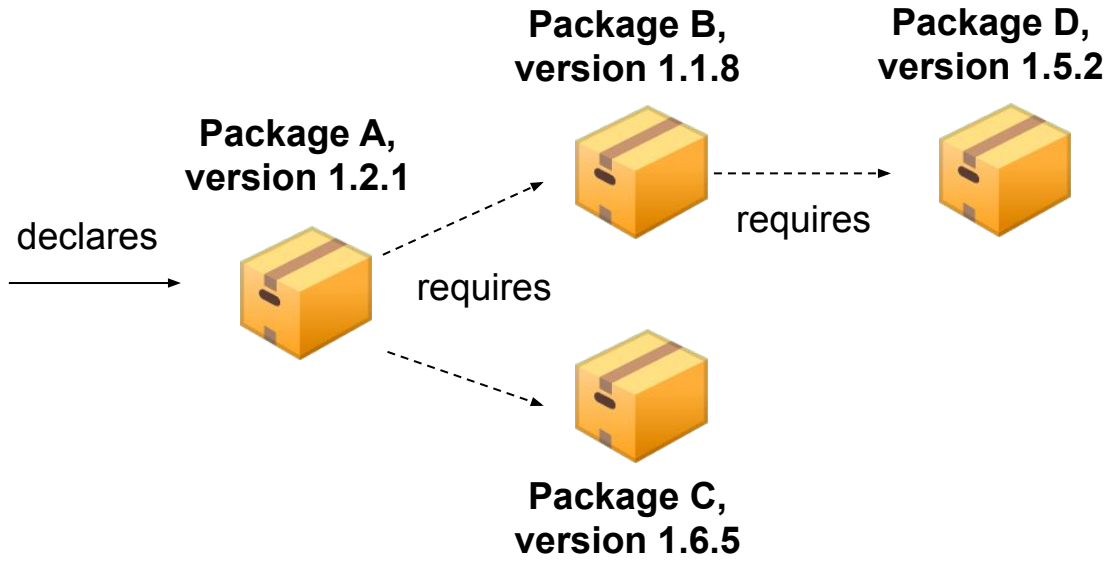
```
module github.com/bmuschko/hello-world

go 1.20

require github.com/spf13/cobra v0.0.3
```



```
module github.com/bmuschko/hello-world

go 1.20

require github.com/spf13/cobra v0.0.5
```

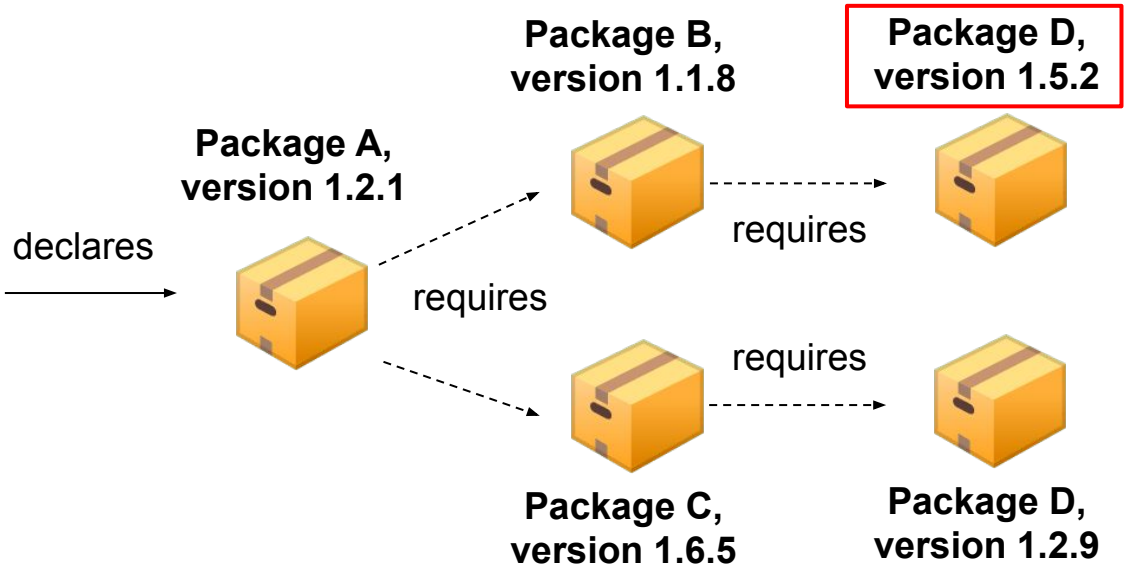
Transitive Dependencies

Modules required by declared dependency to work properly



Minimal Version Selection

“How does the Module system resolve version conflicts?”



Rendering a Dependency Graph

“Show me the declared dependencies and its transitives!”

```
$ go mod graph
github.com/bmuschko/hello-world github.com/spf13/cobra@v0.0.5
github.com/spf13/cobra@v0.0.5 github.com/BurntSushi/toml@v0.3.1
github.com/spf13/cobra@v0.0.5 github.com/cpuguy83/go-md2man@v1.0.10
...
```

aka

```
github.com/bmuschko/hello-world
└─ github.com/spf13/cobra@v0.0.5
    ├─ github.com/BurntSushi/toml@v0.3.1
    └─ github.com/cpuguy83/go-md2man@v1.0.10
```

Identifying the Selected Version

List the dependency versions to get a clearer picture

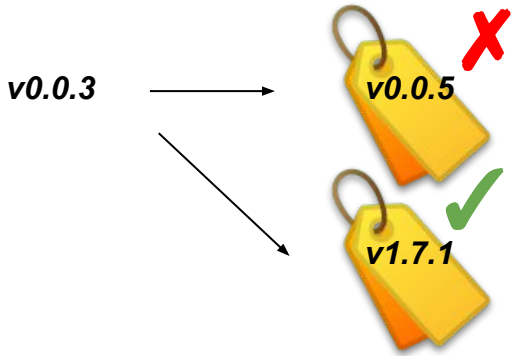
```
$ go list -m all
github.com/bmuschko/hello-world
...
github.com/package-d/v1.5.2
...
```

Module path

Selected package version

Replacing Dependency Versions

“I know better. I want that exact version of a dependency.”

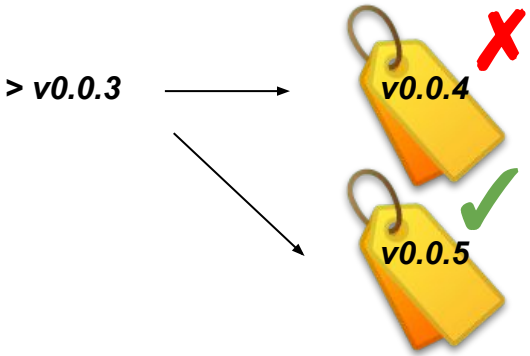



```
module github.com/bmuschko/hello-world

go 1.20

require (
    github.com/spf13/cobra v0.0.3
    github.com/spf13/pflag v1.0.5 // indirect
)

replace github.com/spf13/pflag => github.com/spf13/pflag v1.0.3
```

You can point to module located in VCS or local filesystem

Excluding Dependency Versions

“This specific version has a bug. Definitely don’t select it.”




```
module github.com/bmuschko/hello-world

go 1.20

require (
    github.com/spf13/cobra >v0.0.3
    github.com/spf13/pflag v1.0.5 // indirect
)

exclude github.com/spf13/cobra v0.0.4
```

Selects the next higher version, fails if there’s no better version available

EXERCISE

Handling Transitive Dependencies

Verifying Dependencies

“Do dependencies in cache have expected content?”

```
github.com/spf13/cobra v0.0.5 h1:f0B+LkLX6DtmRH1isoNA9VTtNUK9K8xYd28JNNfOv/s=
github.com/spf13/cobra v0.0.5/go.mod h1:3K3wKZymM7VvHMDS9+Akkh4K60UwM26emMESw8tLCHU=
```

```
$ tree $GOPATH/pkg/mod/github.com
github.com
└── spf13
    ├── cobra@v0.0.5
    │   ├── ...
```

```
$ go mod verify
all modules verified
```

Verifying Dependencies

Identifies malicious changes to dependencies

```
$ cd $GOPATH/pkg/mod/github.com/spf13/cobra@v0.0.5
$ vim cobra.go
// Make changes to content
```

```
$ go mod verify
github.com/spf13/cobra v0.0.5: dir has been modified ↵
(/Users/bmuschko/go/pkg/mod/github.com/spf13/cobra@v0.0.5)
```

Why is a Module Needed?

Doesn't really provide a reason but a pointer to dependency

```
$ go mod why github.com/spf13/pflag
# github.com/spf13/pflag
github.com/bmuschko/hello-world/cmd
github.com/spf13/cobra
github.com/spf13/pflag
```

aka

```
github.com/bmuschko/hello-world/cmd
└─ github.com/spf13/cobra
    └─ github.com/spf13/pflag
```

Removing Unused Dependencies

Expensive operation so you'll have to trigger it on occasion

```
$ go mod tidy
```

```
module github.com/bmuschko/hello-world

go 1.20

require github.com/spf13/cobra v0.0.5
```

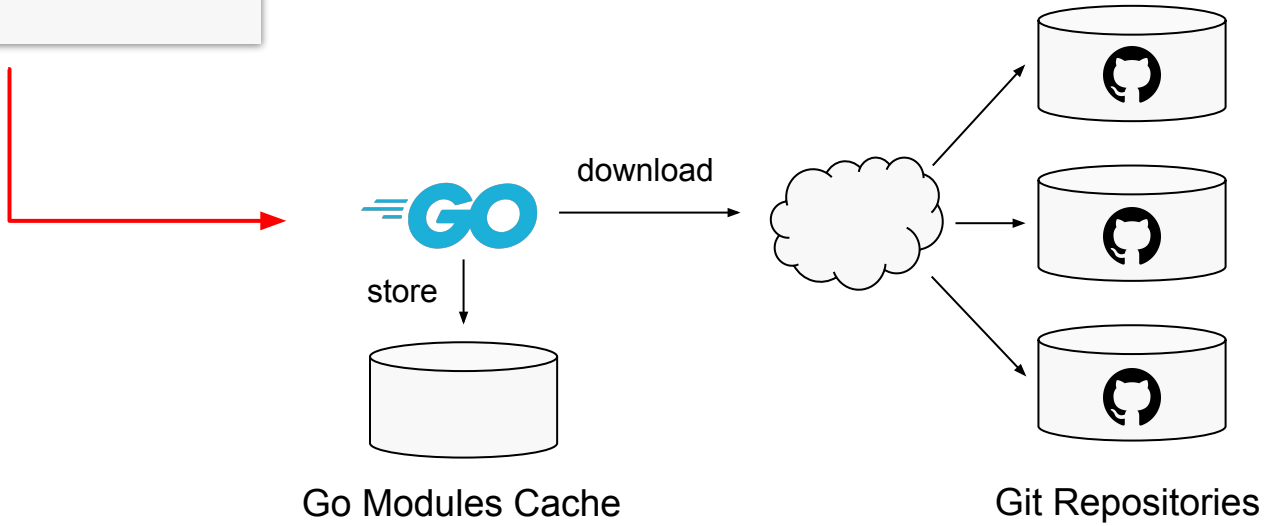
```
module github.com/bmuschko/hello-world

go 1.20
```

Removal of dependencies if not used in the application code anymore

Downloading Dependencies

“Download all dependencies so I can work offline!”

```
$ go mod download
```



EXERCISE

Verifying and Tidying Dependencies

Q & A

Managing Modules Versions

Controlling Version Selection

Module Versioning Rules

“Is there more to it? What are the corner cases?”

- **v1 or earlier** e.g. `v0.0.5` or `v1.2.3`
- **v2 or later** e.g. `v2.6.1` or `v3.4.6`
- **Unversioned modules** e.g. the commit hash `14457a6`

v1 and earlier

Characteristics and import statement

- pre-v1 versions can contain breaking changes
- Version selection rules are based on order of major, minor, patch
- Assumes no breaking changes within this major version range

```
import github.com/spf13/cobra
```

Usage of v1 and earlier

Import statement doesn't require spelling out version

```
package main

import "github.com/bmuschko/calculator"

func main() {
    calculator.Add(1, 2)
}
```

v2 and later

Characteristics and import statement

- Assumes breaking changes between v1 and v2
- If v2+ version, then the import path appends version
- Code can import modules with different major versions

```
import github.com/spf13/cobra/v2
```

Usage of v2 and later

Import statement requires spelling out version

```
package main

import calculatorV2 "github.com/bmuschko/calculator/v2"

func main() {
    calculatorV2.Add(1, 2, 3)
}
```

Using Multiple Versions

Provides convenient migration path for breaking APIs

```
package main

import "github.com/bmuschko/calculator"
import calculatorV2 "github.com/bmuschko/calculator/v2"

func main() {
    calculator.Add(1, 2)
    calculatorV2.Add(1, 2, 3)
}
```

Unversioned Modules

Characteristics and import statement

- Assumes that module hasn't been released or adopted modules yet
- Uses `(v)0.0.0` as semantic version for referencing module
- Needs to point to timestamp and commit hash to identify changeset

```
import github.com/spf13/cobra
```

EXERCISE

Using a Dependency with Multiple Major Versions

Resolving Specific Versions

Identifiers for appropriate use cases (aka Module queries)

- Specific version: `@v1.2.3`
- Latest of a major version: `@v3`
- Wildcard: `@latest` (default if not specified) or `@branch`
- Specific commit: `@14457a6`

Using Version Notations

Works when invoking `go get` and in `go.mod`

```
$ go get github.com/spf13/cobra @master
```

go.mod

```
require github.com/spf13/cobra v0
```

```
require github.com/spf13/cobra v0.0.5
```

IDE will automatically resolve version and replace value

Semantic Version Comparison

Version selection follows “closest match wins”

Greater Than Equals

```
@">=v1.5.2"
```

Available Versions

```
v1.1.0
v1.2.0
v1.5.2
v1.6.1
v2.5.1
```

Less Than

```
@"<v2.4.1"
```

Available Versions

```
v1.1.0
v1.2.0
v1.5.2
v1.6.1
v2.5.1
```

Greater than equals 1.5.2 **and** less than 2.4.1 is not possible

EXERCISE

Executing Module Queries

Q & A

BREAK

Advanced Modules Techniques

Vendoring, Proxies and Migration Strategies to Modules

Vendoring...Why?

“I don’t want to download dependencies when building.”

- Deliver source code including dependencies to customer
- No access to internet or Modules proxy
- Backward compatibility or migrating away from Modules concept

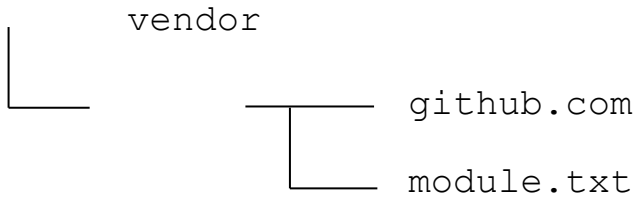
https://groups.google.com/forum/#!msg/golang-dev/FTMScX1fsYk/uEUSjBAHAwAJ

Vendoring Dependencies

Download dependencies and copy them in vendor directory

```
$ go mod vendor
```

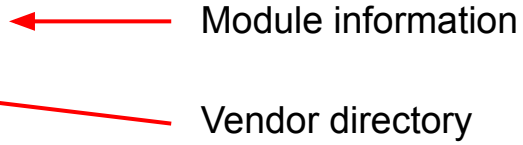
```
module github.com/bmuschko/hello-world

go 1.20

require github.com/spf13/cobra v0.0.5
```



Vendoring Text File

Mapping of Module information to vendor directory

```
# github.com/inconshreveable/mousetrap v1.0.0
github.com/inconshreveable/mousetrap
# github.com/spf13/cobra v0.0.5
github.com/spf13/cobra
# github.com/spf13/pflag v1.0.3
github.com/spf13/pflag
```

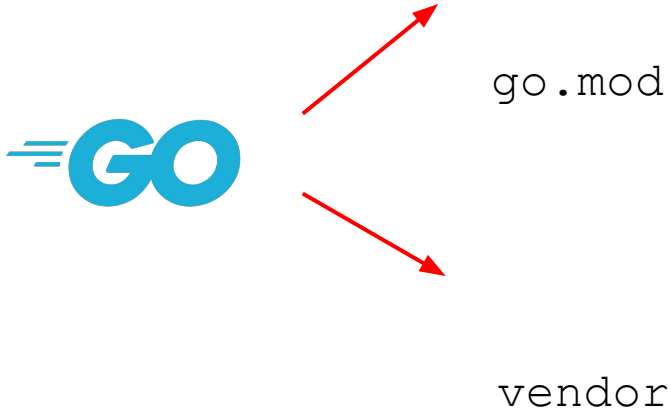


module.txt

Using Vendored Dependencies

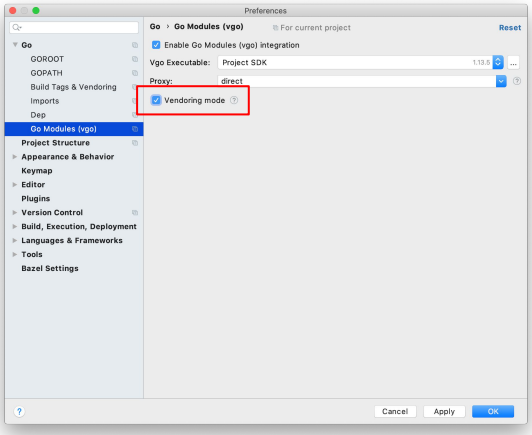
Vendoring has to be enabled explicitly

```
$ go build -mod=vendor
```



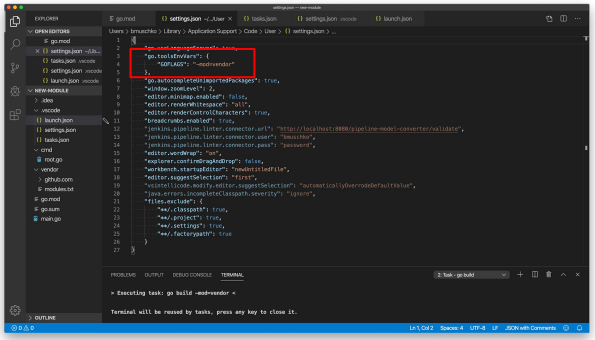
Vendoring in IDEs

Requires enabling the feature



GoLand

```
"go.toolsEnvVars": {
    "GOFLAGS": "-mod=vendor"
}
```



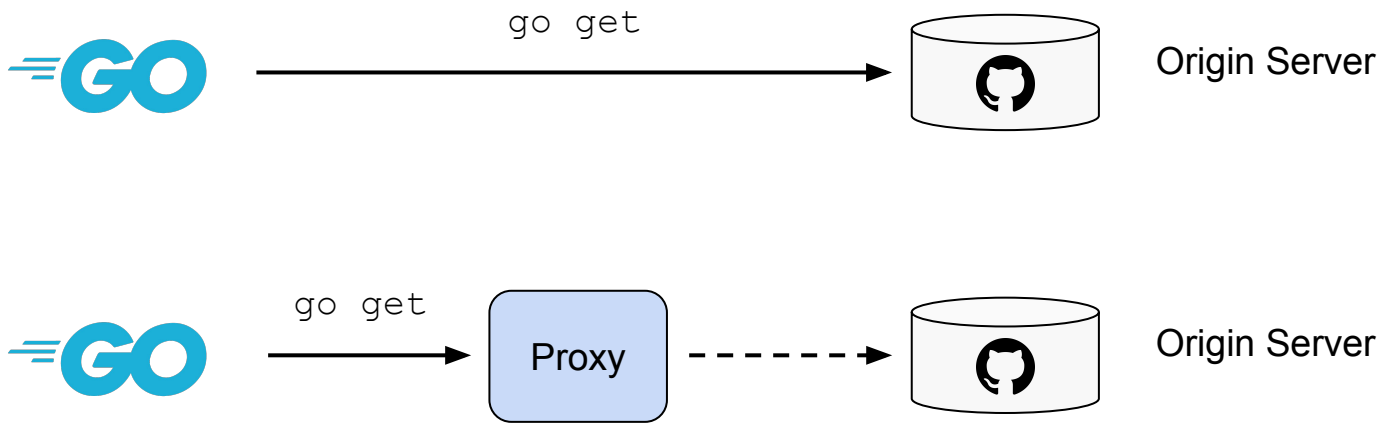
VSCode

EXERCISE

Vendoring
Dependencies

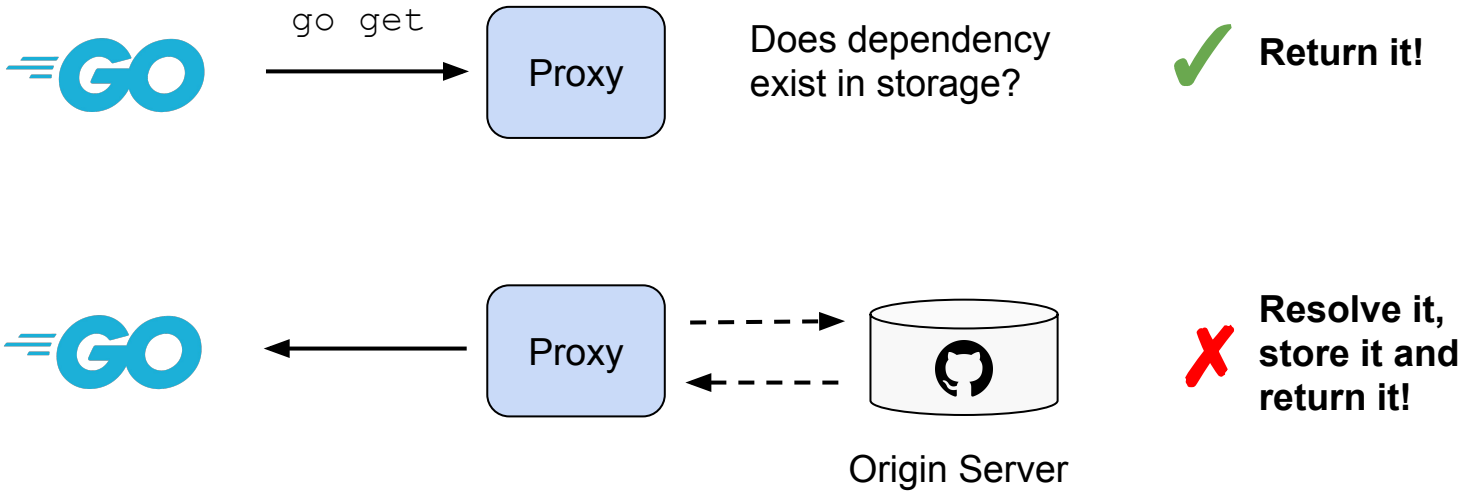
Proxy for Modules

Improve reliability, performance and security



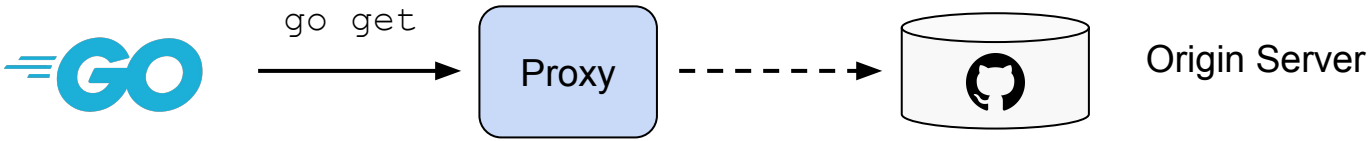
Proxy Resolution Workflow

Only talk to origin server if dependency doesn't exist yet



Default Google Proxy

Go module mirror configured by default

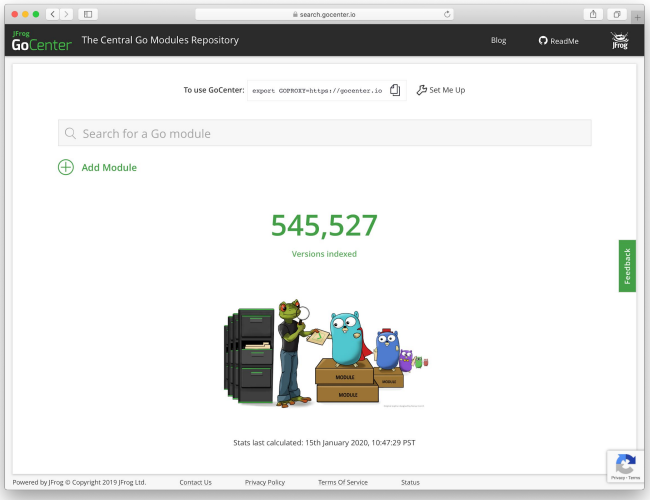


`https://proxy.golang.org/`

Opt-out with `GOPROXY=direct`

SaaS Modules Proxy

Cache for Modules hosted and maintained by JFrog



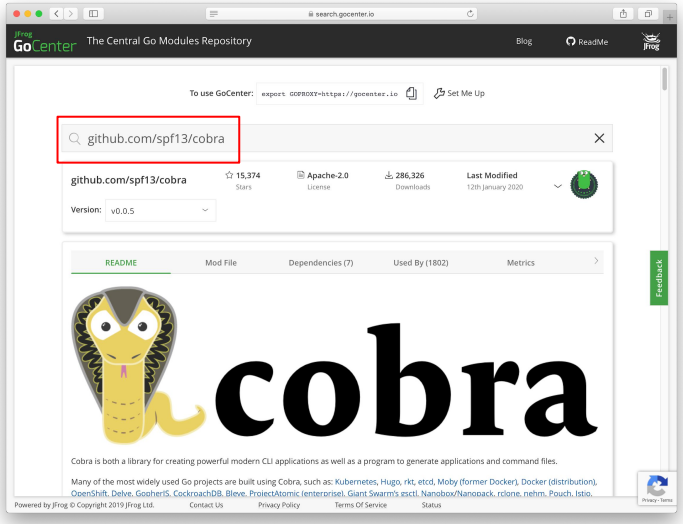
JFrog GoCenter

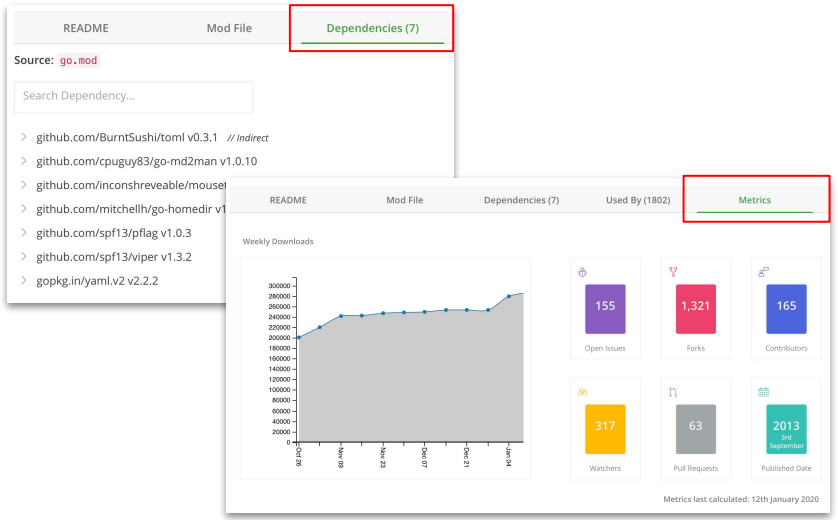
https://search.gocenter.io/

```
export GOPROXY=https://gocenter.io
```

Dependency in GoCenter

Deprecated since March 31st, 2021





Self-Hosted Proxy

“I don’t want to reach out to the internet!”

Commercial

https://jfrog.com/artifactory/

Open Source

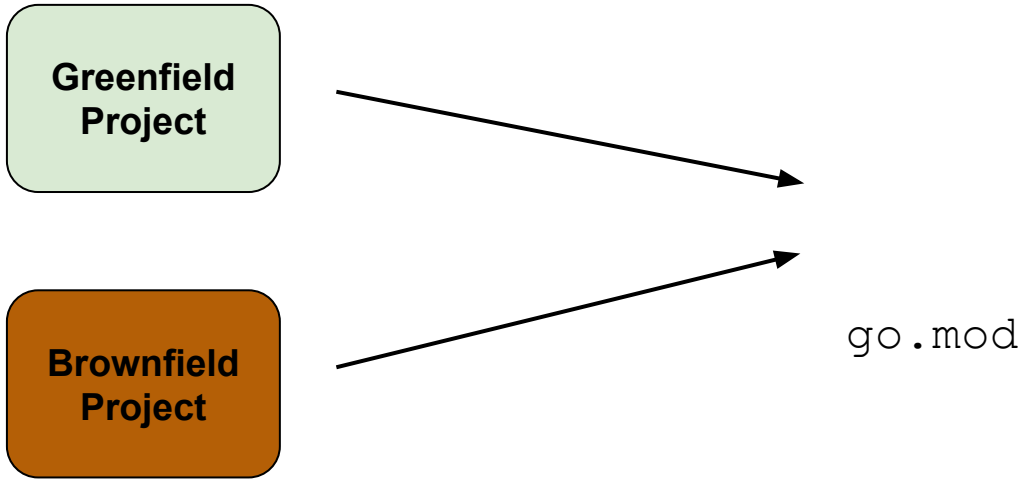
https://github.com/gomods/athens

EXERCISE

Using JFrog GoCenter
To Proxy Dependencies

Migration Strategies

Go Modules are here to stay, better get used to it...



Starting From Scratch

No dependency metadata available

- Start by running `go mod init` to start project
- Run `go mod tidy` to derive module information from imports (will use latest version)
- Modify versions in `go.mod` as needed

Using Another Package Manager

Dependency information exists and may be derived

- Start by running `go mod init` to translate dependency information
- Edit import statements if you use dependencies with v2+
- Run `go mod tidy` to clean up dependency definitions
- Delete metadata of “old” package manager

Existing Vendoring Information

Dependencies available in vendor directory

- Initialize the project with `go mod init`
- Use the `-mod=vendor` CLI flag for go commands
- Can’t extract vendor information into `go.mod`
- Try to derive dependency information and fill into `go.mod`

EXERCISE

Migrating to Go Modules

Q & A

Summary

Let’s wrap up what we’ve learnt...

O'REILLY®

Thank you