JANUARY 2026 SERIES

FROM GO BUILD TO GO RUN

GOLANG 2026 - NIV RAVE

#27

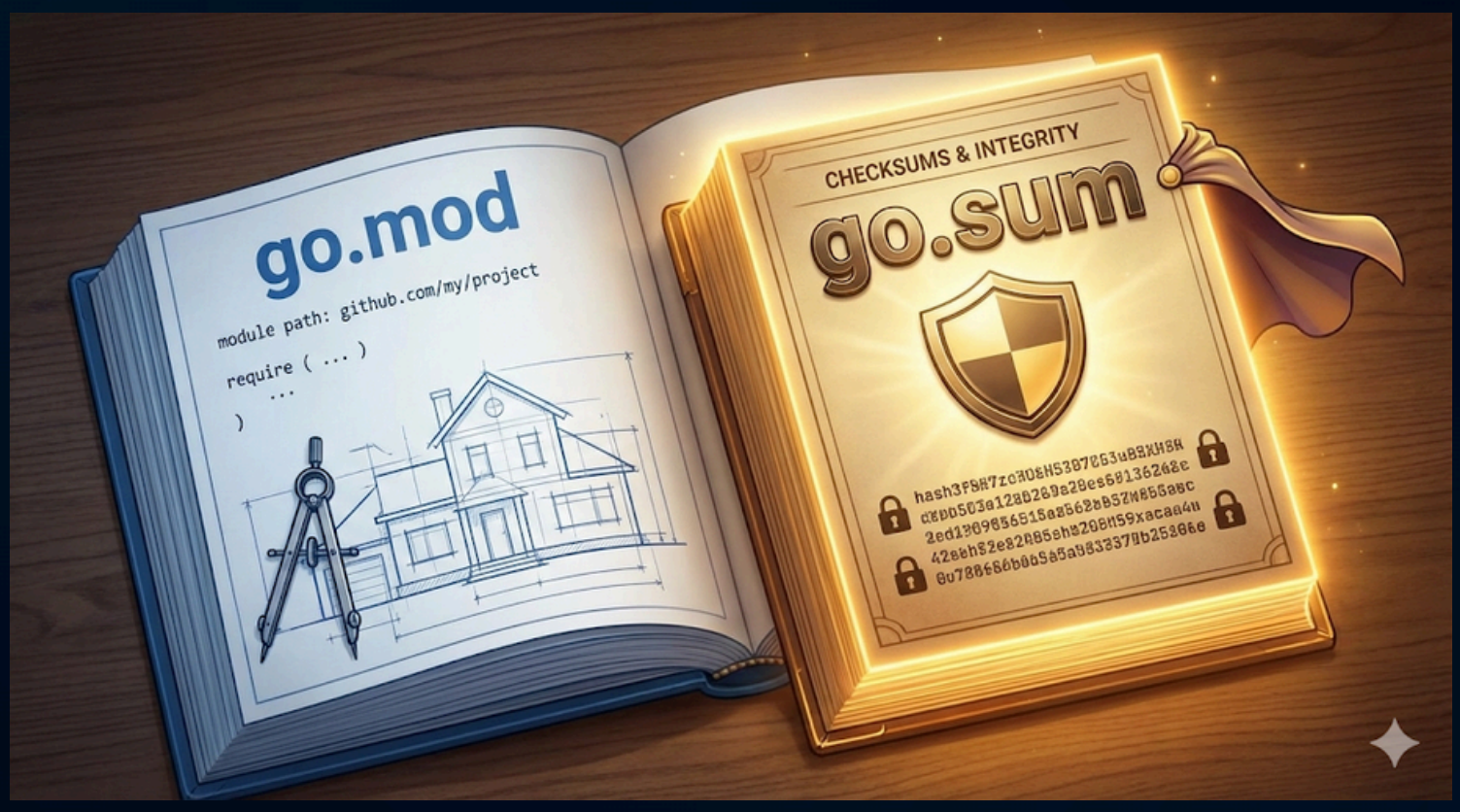
PACKAGES AND GO MOD

BEYOND GO GET

Reproducibility by Default

The Hermetic Build

The *go.mod* file defines the module's path and its dependency requirements. But the real hero is *go.sum*.



go.sum is not a lockfile in the traditional sense (*like package-lock.json*). It is a list of cryptographic hashes. If a dependency's code changes on the proxy, *go build* will fail, protecting you from "invisible" upstream mutations.

The internal Directory Pattern

Internal vs. External Packages

Go has a built-in access modifier at the file system level. Any package inside a directory named *internal* can only be imported by code within the same parent tree.

The Structure:

```
my-project/
├── internal/
│   └── auth/       // Only usable by my-project
├── pkg/
│   └── util/       // Usable by anyone (public API)
└── go.mod
```

Use *internal* aggressively. It allows you to refactor without breaking external consumers, keeping your public API surface area small and manageable.

Minimal Version Selection

Understanding MVS: Stability by Design

Most package managers (like npm or Cargo) default to the newest compatible version. Go takes the opposite approach. It selects the minimal (oldest) version that satisfies all requirements in your dependency graph.

- The Problem it Solves: "Version Creep." If you haven't explicitly asked for a newer version of a library, Go won't force it on you just because it exists. This makes your builds predictable and less likely to break due to an upstream update you didn't vet.

The Logic:

1. **Requirement** A: Needs *library-v1.2*.
2. **Requirement** B: Needs *library-v1.5*.
3. **Result**: Go selects v1.5 because it is the minimal version that satisfies both. It will not jump to v1.9 even if it's available, unless you explicitly run *go get -u*.

MVS shifts the responsibility of upgrading to the developer. This is a "feature," not a bug. It forces you to be intentional about updates. If you need a security patch in v1.5.2, you must explicitly bump it. This prevents the "**it worked on my machine but failed in CI**" syndrome caused by floating dependency versions.

Keeping the House Clean

The *go mod tidy* Discipline

go mod tidy is the only way to ensure your *go.mod* and *go.sum* files accurately reflect your source code.

```
# Cleans unused deps and adds missing ones
go mod tidy

# Verifies that dependencies haven't been tampered with
go mod verify
```

Senior Take: Run go mod tidy in your CI/CD pipeline. If the check results in a diff, fail the build. Your repo should never have "ghost" dependencies.

Goproxy and Security

The Corporate Firewall

By default, Go fetches modules from proxy.golang.org. For private company code, this won't work.

```
# Tell Go to skip the proxy for internal gitlab/github
export GOPRIVATE=github.com/my-company/*
```

In highly regulated environments, consider a private proxy like Athens. This ensures that even if a developer deletes a GitHub repo, your builds remain intact.

To pkg or not to pkg?

Why *pkg* is controversial

You'll see many repos using a */pkg* folder for public code. It is not a Go standard, but a community convention.

The Trade-off:
Pro: Keeps the root directory clean.
Con: Adds an extra nesting level in every import path (import "github.com/user/repo/pkg/util").

Recommendation: For small libraries, put code in the root. For large monorepos, use */internal* and */pkg* to separate concerns.

Graphing Dependencies

Visualizing the Web

Complex projects lead to "Dependency Hell."

Go provides tools to visualize the graph.

```
# See why a specific package is being pulled in
go mod why -m github.com/google/uuid

# Generate a graph (pipe to graphviz)
go mod graph
```

GOLANG /// BA

Summary:

- Use *internal* to hide implementation details.
- *go.sum* is your security guard.
- MVS favors stability over "latest and greatest."
- Use *GOPRIVATE* for internal infrastructure.

Do you prefer the flat structure (all code in root) or the /pkg and /internal layout for your services? Let's discuss the architectural merits.👇