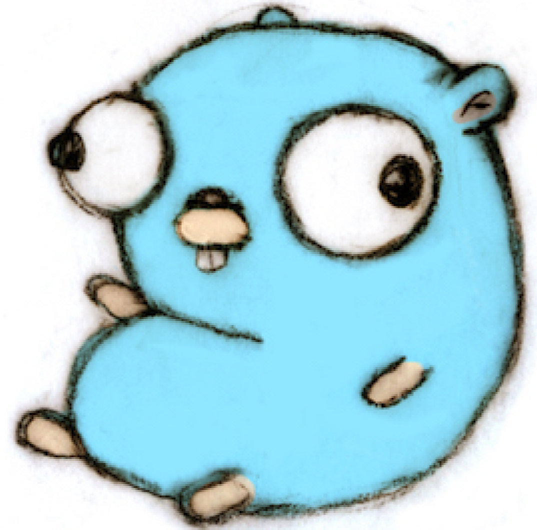
Thirteen Years of Go

Russ Cox, for the Go team
10 November 2022

Today we celebrate the thirteenth birthday of the Go open source release. The Gopher is a teenager!



It's been an eventful year for Go. The most significant event was the release of Go 1.18 in March, which brought many improvements but most notably Go workspaces, fuzzing, and generics.

Workspaces make it easy to work on multiple modules simultaneously, which is most helpful when you are maintaining a set of related modules with module dependencies between them. To learn about workspaces, see Beth Brown's blog post "Get familiar with workspaces" and the workspace reference.

Fuzzing is a new feature of `go test` that helps you find inputs that your code doesn't handle properly: you define a fuzz test that should pass for any input at all, and then fuzzing tries different random inputs, guided by code coverage, to try to make the fuzz test fail. Fuzzing is particularly useful when developing code that must be robust against arbitrary (even attacker-controlled) inputs. To learn more about fuzzing, see the tutorial "Getting started with fuzzing" and the fuzzing reference, and keep an eye out for Katie Hockman's GopherCon 2022 talk "Fuzz Testing Made Easy", which should be online soon.

Generics, quite possibly Go's most requested feature, adds parametric polymorphism to Go, to allow writing code that works with a variety of different types but is still statically checked at compile time. To learn more about generics, see the tutorial "Getting started with generics". For more detail see the blog posts "An Introduction to Generics" and "When to Use Generics", or the talks "Using Generics in Go" from Go Day on Google Open Source Live 2021, and "Generics!" from GopherCon 2021, by Robert Griesemer and Ian Lance Taylor.

Compared to Go 1.18, the Go 1.19 release in August was a relatively quiet one: it focused on refining and improving the features that Go 1.18 introduced as well as internal stability improvements and optimizations. One visible change in Go 1.19 was the addition of support for links, lists, and headings in Go doc comments. Another was the addition of a soft memory limit for the garbage collector, which is particularly useful in container workloads. For more about recent garbage collector improvements, see Michael Knyszek's blog post "Go runtime: 4 years later", his talk "Respecting Memory Limits in Go", and the new "Guide to the Go Garbage Collector".

We've continued to work on making Go development scale gracefully to ever larger code bases, especially in our work on VS Code Go and the Gopls language server. This year, Gopls releases focused on improving stability and performance, while delivering support for generics as well as new analyses and code lenses. If

you aren't using VS Code Go or Gopls yet, give them a try. See Suzy Mueller's talk "Building Better Projects with the Go Editor" for an overview. And as a bonus, Debugging Go in VS Code got more reliable and powerful with Delve's native Debug Adapter Protocol support. Try Suzy's "Debugging Treasure Hunt"!

Another part of development scale is the number of dependencies in a project. A month or so after Go's 12th birthday, the Log4shell vulnerability served as a wake-up call for the industry about the importance of supply chain security. Go's module system was designed specifically for this purpose, to help you understand and track your dependencies, identify which specific ones you are using, and determine whether any of them have known vulnerabilities. Filippo Valsorda's blog post "How Go Mitigates Supply Chain Attacks" gives an overview of our approach. In September, we previewed Go's approach to vulnerability management in Julie Qiu's blog post "Vulnerability Management for Go". The core of that work is a new, curated vulnerability database and a new govulncheck command, which uses advanced static analysis to eliminate most of the false positives that would result from using module requirements alone.

Part of our effort to understand Go users is our annual end-of-year Go survey. This year, our user experience researchers added a lightweight mid-year Go survey as well. We aim to gather enough responses to be statistically significant without being a burden on the Go community as a whole. For the results, see Alice Merrick's blog post "Go Developer Survey 2021 Results" and Todd Kulesza's post "Go Developer Survey 2022 Q2 Results".

As the world starts traveling more, we've also been happy to meet many of you in person at Go conferences in 2022, particularly at GopherCon Europe in Berlin in July and at GopherCon in Chicago in October. Last week we held our annual virtual event, Go Day on Google Open Source Live. Here are some of the talks we've given at those events:

- "How Go Became its Best Self", by Cameron Balahan, at GopherCon Europe.
- "Go team Q&A", with Cameron Balahan, Michael Knyszek, and Than McIntosh, at GopherCon Europe.
- "Compatibility: How Go Programs Keep Working", by Russ Cox at GopherCon.
- "A Holistic Go Experience", by Cameron Balahan at GopherCon (video not yet posted)
- "Structured Logging for Go", by Jonathan Amsterdam at Go Day on Google Open Source Live
- "Writing your Applications Faster and More Securely with Go", by Cody Oss at Go Day on Google Open Source Live
- "Respecting Memory Limits in Go, by Michael Knyszek at Go Day on Google Open Source Live

One other milestone for this year was the publication of "The Go Programming Language and Environment", by Russ Cox, Robert Griesemer, Rob Pike, Ian Lance Taylor, and Ken Thompson, in *Communications of the ACM*. The article, by the original designers and implementers of Go, explains what we believe makes Go so popular and productive. In short, it is that Go effort focuses on delivering a full development environment targeting the entire software development process, with a focus on scaling both to large software engineering efforts and large deployments.

In Go's 14th year, we'll keep working to make Go the best environment for software engineering at scale. We plan to focus particularly on supply chain security, improved compatibility, and structured logging, all of which have been linked already in this post. And there will be plenty of other improvements as well, including profile-guided optimization.

Thank You!

Go has always been far more than what the Go team at Google does. Thanks to all of you—our contributors and everyone in the Go community—for your help making Go the successful programming environment that it is today. We wish you all the best in the coming year.

Previous article: Go runtime: 4 years later
Blog Index