
xgo - Go CGO cross compiler

Although Go strives to be a cross platform language, cross compilation from one platform to another is not as simple as it could be, as you need the Go sources bootstrapped to each platform and architecture.

The first step towards cross compiling was Dave Cheney's `golang-crosscompile` package, which automatically bootstrapped the necessary sources based on your existing Go installation. Although this was enough for a lot of cases, certain drawbacks became apparent where the official libraries used CGO internally: any dependency to third party platform code is unavailable, hence those parts don't cross compile nicely (native DNS resolution, system certificate access, etc).

A step forward in enabling cross compilation was Alan Shreve's `gonative` package, which instead of bootstrapping the different platforms based on the existing Go installation, downloaded the official pre-compiled binaries from the `golang` website and injected those into the local toolchain. Since the pre-built binaries already contained the necessary platform specific code, the few missing dependencies were resolved, and true cross compilation could commence... of pure Go code.

However, there was still one feature missing: cross compiling Go code that used CGO itself, which isn't trivial since you need access to OS specific headers and libraries. This becomes very annoying when you need access only to some trivial OS specific functionality (e.g. query the CPU load), but need to configure and maintain separate build environments to do it.

Enter xgo

My solution to the challenge of cross compiling Go code with embedded C/C++ snippets (i.e. `CGO_ENABLED=1`) is based on the concept of lightweight Linux containers. All the necessary Go tool-chains, C cross compilers and platform headers/libraries have been assembled into a single Docker container, which can then be called as if a single command to compile a Go package to various platforms and architectures.

Installation

Although you could build the container manually, it is available as an automatic trusted build from Docker's container registry (not insignificant in size):

```
1 docker pull karalabe/xgo-latest
```

To prevent having to remember a potentially complex Docker command every time, a lightweight Go wrapper was written on top of it.

```
1 go get github.com/karalabe/xgo
```

Usage

Simply specify the import path you want to build, and xgo will do the rest:

```
1 $ xgo github.com/project-iris/iris
2 ...
3
4 $ ls -al
5 -rwxr-xr-x 1 root root 9995000 Nov 24 16:44 iris-android-16-arm
6 -rwxr-xr-x 1 root root 6776500 Nov 24 16:44 iris-darwin-10.6-386
7 -rwxr-xr-x 1 root root 8755532 Nov 24 16:44 iris-darwin-10.6-amd64
8 -rwxr-xr-x 1 root root 7114176 Nov 24 16:45 iris-ios-5.0-arm
9 -rwxr-xr-x 1 root root 10135248 Nov 24 16:44 iris-linux-386
10 -rwxr-xr-x 1 root root 12598472 Nov 24 16:44 iris-linux-amd64
11 -rwxr-xr-x 1 root root 10040464 Nov 24 16:44 iris-linux-arm
12 -rwxr-xr-x 1 root root 7516368 Nov 24 16:44 iris-windows-4.0-386.
    exe
13 -rwxr-xr-x 1 root root 9549416 Nov 24 16:44 iris-windows-4.0-amd64
    .exe
```

If the path is not a canonical import path, but rather a local path (starts with a dot `.` or a dash `/`), xgo will use the local GOPATH contents for the cross compilation.

Build flags

A handful of flags can be passed to `go build`. The currently supported ones are

- `-v`: prints the names of packages as they are compiled
- `-x`: prints the build commands as compilation progresses
- `-race`: enables data race detection (supported only on amd64, rest built without)
- `-tags='tag list'`: list of build tags to consider satisfied during the build
- `-ldflags='flag list'`: arguments to pass on each go tool link invocation
- `-buildmode=mode`: binary type to produce by the compiler

Go releases

As newer versions of the language runtime, libraries and tools get released, these will get incorporated into xgo too as extensions layers to the base cross compilation image (only Go 1.3 and above will be supported).

You can select which Go release to work with through the `-go` command line flag to `xgo` and if the specific release was already integrated, it will automatically be retrieved and installed.

```
1 $ xgo -go 1.6.1 github.com/project-iris/iris
```

Additionally, a few wildcard release strings are also supported:

- `latest` will use the latest Go release (this is the default)
- `1.6.x` will use the latest point release of a specific Go version
- `1.6-develop` will use the develop branch of a specific Go version
- `develop` will use the develop branch of the entire Go repository

Output prefixing

`xgo` by default uses the name of the package being cross compiled as the output file prefix. This can be overridden with the `-out` flag.

```
1 $ xgo -out iris-v0.3.2 github.com/project-iris/iris
2 ...
3
4 $ ls -al
5 -rwxr-xr-x  1 root  root   9995000 Nov 24 16:44 iris-v0.3.2-android-16-
   arm
6 -rwxr-xr-x  1 root  root   6776500 Nov 24 16:44 iris-v0.3.2-darwin
   -10.6-386
7 -rwxr-xr-x  1 root  root   8755532 Nov 24 16:44 iris-v0.3.2-darwin
   -10.6-amd64
8 -rwxr-xr-x  1 root  root   7114176 Nov 24 16:45 iris-v0.3.2-ios-5.0-arm
9 -rwxr-xr-x  1 root  root  10135248 Nov 24 16:44 iris-v0.3.2-linux-386
10 -rwxr-xr-x  1 root  root  12598472 Nov 24 16:44 iris-v0.3.2-linux-amd64
11 -rwxr-xr-x  1 root  root  10040464 Nov 24 16:44 iris-v0.3.2-linux-arm
12 -rwxr-xr-x  1 root  root   7516368 Nov 24 16:44 iris-v0.3.2-windows
   -4.0-386.exe
13 -rwxr-xr-x  1 root  root   9549416 Nov 24 16:44 iris-v0.3.2-windows
   -4.0-amd64.exe
```

Branch selection

Similarly to `go get`, `xgo` also uses the `master` branch of a repository during source code retrieval. To switch to a different branch before compilation pass the desired branch name through the `--branch` argument.

```
1 $ xgo --branch release-branch.gol.4 golang.org/x/tools/cmd/goimports
2 ...
3
```

```
4 $ ls -al
5 -rwxr-xr-x 1 root root 4171248 Nov 24 16:40 goimports-android-16-
   arm
6 -rwxr-xr-x 1 root root 4139868 Nov 24 16:40 goimports-darwin
   -10.6-386
7 -rwxr-xr-x 1 root root 5186720 Nov 24 16:40 goimports-darwin-10.6-
   amd64
8 -rwxr-xr-x 1 root root 3202364 Nov 24 16:40 goimports-ios-5.0-arm
9 -rwxr-xr-x 1 root root 4189456 Nov 24 16:40 goimports-linux-386
10 -rwxr-xr-x 1 root root 5264136 Nov 24 16:40 goimports-linux-amd64
11 -rwxr-xr-x 1 root root 4209416 Nov 24 16:40 goimports-linux-arm
12 -rwxr-xr-x 1 root root 4348416 Nov 24 16:40 goimports-windows
   -4.0-386.exe
13 -rwxr-xr-x 1 root root 5415424 Nov 24 16:40 goimports-windows-4.0-
   amd64.exe
```

Remote selection

Yet again similarly to `go get`, `xgo` uses the repository remote corresponding to the import path being built. To switch to a different remote while preserving the original import path, use the `--remote` argument.

```
1 $ xgo --remote github.com/golang/tools golang.org/x/tools/cmd/goimports
2 ...
```

Package selection

If you used the above *branch* or *remote* selection mechanisms, it may happen that the path you are trying to build is only present in the specific branch and not the default repository, causing Go to fail at locating it. To circumvent this, you may specify only the repository root for `xgo`, and use an additional `--pkg` parameter to select the exact package within, honoring any prior *branch* and *remote* selections.

```
1 $ xgo --pkg cmd/goimports golang.org/x/tools
2 ...
3
4 $ ls -al
5 -rwxr-xr-x 1 root root 4194956 Nov 24 16:38 goimports-android-16-
   arm
6 -rwxr-xr-x 1 root root 4164448 Nov 24 16:38 goimports-darwin
   -10.6-386
7 -rwxr-xr-x 1 root root 5223584 Nov 24 16:38 goimports-darwin-10.6-
   amd64
8 -rwxr-xr-x 1 root root 3222848 Nov 24 16:39 goimports-ios-5.0-arm
9 -rwxr-xr-x 1 root root 4217184 Nov 24 16:38 goimports-linux-386
```

10	-rwxr-xr-x	1	root	root	5295768	Nov	24	16:38	goimports-linux-amd64
11	-rwxr-xr-x	1	root	root	4233120	Nov	24	16:38	goimports-linux-arm
12	-rwxr-xr-x	1	root	root	4373504	Nov	24	16:38	goimports-windows
									-4.0-386.exe
13	-rwxr-xr-x	1	root	root	5450240	Nov	24	16:38	goimports-windows-4.0-
									amd64.exe

This argument may at some point be integrated into the import path itself, but for now it exists as an independent build parameter. Also, there is not possibility for now to build multiple commands in one go.

Limit build targets

By default `xgo` will try and build the specified package to all platforms and architectures supported by the underlying Go runtime. If you wish to restrict the build to only a few target systems, use the comma separated `--targets` CLI argument:

- `--targets=linux/arm`: builds only the ARMv5 Linux binaries (`arm-6/arm-7` allowed)
- `--targets=windows/*,darwin/*`: builds all Windows and OSX binaries
- `--targets=*/arm`: builds ARM binaries for all platforms
- `--targets=/*/*`: builds all supported targets (default)

The supported targets are:

- Platforms: `android`, `darwin`, `ios`, `linux`, `windows`
- Achitectures: `386`, `amd64`, `arm-5`, `arm-6`, `arm-7`, `arm64`, `mips`, `mipsle`, `mips64`, `mips64le`

Platform versions

By default `xgo` tries to cross compile to the lowest possible versions of every supported platform, in order to produce binaries that are portable among various versions of the same operating system. This however can lead to issues if a used dependency is only supported by more recent systems. As such, `xgo` supports the selection of specific platform versions by appending them to the OS target string.

- `--targets=ios-8.1/*`: cross compile to iOS 8.1
- `--targets=android-16/*`: cross compile to Android Jelly Bean
- `--targets=darwin-10.9/*`: cross compile to Mac OS X Mavericks
- `--targets=windows-6.0/*`: cross compile to Windows Vista

The supported platforms are:

-
- All Android APIs up to Android Lollipop 5.0 (API level ids)
 - All Windows APIs up to Windows 8.1 limited by [mingw-w64](#) (API level ids)
 - OSX APIs in the range of 10.6 - 10.11
 - All iOS APIs up to iOS 9.3

Mobile libraries

Apart from the usual runnable binaries, [xgo](#) also supports building library archives for Android ([android/aar](#)) and iOS ([ios/framework](#)). Opposed to [gomobile](#) however [xgo](#) does not derive library APIs from the Go code, so proper CGO C external methods must be defined within the package.

In the case of Android archives, all architectures will be bundled that are supported by the requested Android platform version. For iOS frameworks [xgo](#) will bundle armv7 and arm64 by default, and also the x86_64 simulator builds if the iPhoneSimulator.sdk was injected by the user:

- Create a new docker image based on xgo: [FROM karalabe/xgo-latest](#)
- Inject the simulator SDK: [ADD iPhoneSimulator9.3.sdk.tar.xz /iPhoneSimulator9.3.sdk.tar.xz](#)
- Bootstrap the simulator SDK: [\\$UPDATE_IOS /iPhoneSimulator9.3.sdk.tar.xz](#)

CGO dependencies

The main differentiator of [xgo](#) versus other cross compilers is support for basic embedded C/C++ code and target-platform specific OS SDK availability. The current [xgo](#) release introduces an experimental CGO *dependency* cross compilation, enabling building Go programs that require external C/C++ libraries.

It is assumed that the dependent C/C++ library is [configure/make](#) based, was properly prepared for cross compilation and is available as a tarball download ([.tar](#), [.tar.gz](#) or [.tar.bz2](#)). Further plans include extending this to cmake based projects, if need arises (please open an issue if it's important to you).

Such dependencies can be added via the [--deps](#) argument. They will be retrieved prior to starting the cross compilation and the packages cached to save bandwidth on subsequent calls.

A complex sample for such a scenario is building the Ethereum CLI node, which has the GNU Multiple Precision Arithmetic Library as it's dependency.

```
1 $ xgo --deps=https://gmplib.org/download/gmp/gmp-6.1.0.tar.bz2 \  
2   --targets=windows/* github.com/ethereum/go-ethereum/cmd/geth
```

```
3 ...
4
5 $ ls -al
6 -rwxr-xr-x 1 root root 16315679 Nov 24 16:39 geth-windows-4.0-386.exe
7 -rwxr-xr-x 1 root root 19452036 Nov 24 16:38 geth-windows-4.0-amd64.exe
```

Some trivial arguments may be passed to the dependencies' configure script via `--depsargs`.

```
1 $ xgo --deps=https://gmplib.org/download/gmp/gmp-6.1.0.tar.bz2 \
2     --targets=ios/* --depsargs=--disable-assembly \
3     github.com/ethereum/go-ethereum/cmd/geth
4 ...
5
6 $ ls -al
7 -rwxr-xr-x 1 root root 14804160 Nov 24 16:32 geth-ios-5.0-arm
```

Note, that since xgo needs to cross compile the dependencies for each platform and architecture separately, build time can increase significantly.