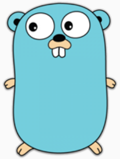
Programming in Go

Matt Holiday
Christmas 2020



Building for Distribution

Go build tools

We've been using `go run` or maybe `go test` to run programs

Now it's time to distribute

- `go build` makes a binary
- `go install` makes one and copies it to `$GOPATH/bin`

Pure Go

We can build “pure” Go programs (with some cautions):

```
$ CGO_ENABLED=0 go build -a -tags netgo,osusergo \
                -ldflags "-extldflags '-static' -s -w" \
                -o lister .
```

Here we must tell Go we’re going to use pure Go networking

A “pure” Go program can put it into a “from-scratch” container

```
$ ldd lister
    not a dynamic executable
```

Go build platforms

Go can cross-compile, too

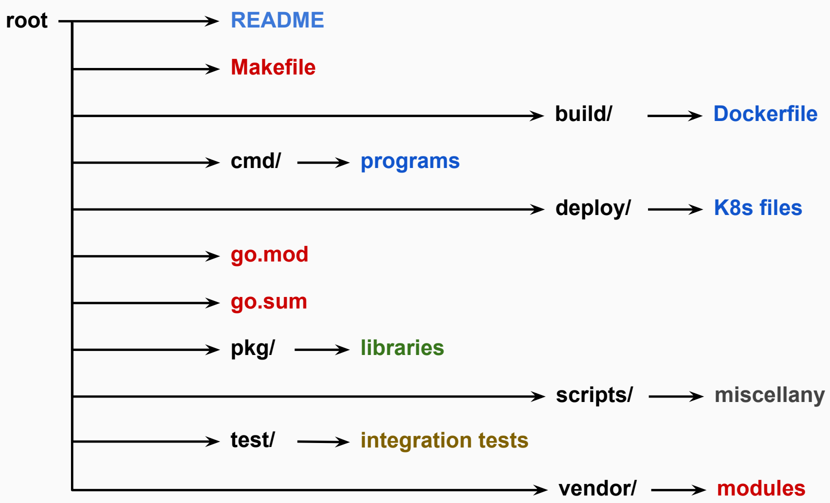
- `$GOARCH` defines the architecture (e.g., `amd64` or `arm64`)
- `$GOOS` defines the operating system (e.g., `linux` or `darwin`)
- `$GOARM` for the ARM chip version (v7, etc.)

We can build for the Raspberry Pi

```
$ GOOS=linux GOARCH=arm GOARM=7 CGO_ENABLED=0 \
                go build -a -tags netgo,osusergo \
                -ldflags "-extldflags '-static' -w" \
                -o mainPi ./main.go

$ file mainPi
mainPi: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV),
        statically linked, not stripped
```

Project layout



Documentation

Your `README.md` file should talk about (among other things)

- overview — who and what is it for?
- developer setup
- project & directory structure
- dependency management
- how to build and/or install it (make targets, etc.)
- how to test it (UTs, integration, end-to-end, load, etc.)
- how to run it (locally, in Docker, etc.)
- database & schema
- credentials & security
- debugging monitoring (metrics, logs)
- CLI tools and their usage

Makefiles

Reasons we might want a `Makefile`

- we need to calculate parameters
- we have other steps and/or dependencies
- because the options are waaaay to long to type
- and we may have non-Go commands (Docker, cloud provider, etc.)

Versioning the executable

In the main program code:

```
// MUST BE SET by go build -ldflags "-X main.version=999"
// like 0.6.14-0-g26fe727 or 0.6.14-2-g9118702-dirty

var version string  // do not remove or modify
```

See Setting compile-time variables for versioning

From the makefile:

```
VERSION=$(shell git describe --tags --long --dirty 2>/dev/null)
BRANCH=$(shell git rev-parse --abbrev-ref HEAD)

xyz: $(SOURCES)
	go build -mod=vendor -ldflags "-X main.version=$(VERSION)" -o $@ ./cmd/xyz
```

Building in Docker

We can use Docker to build as well as run

- multi-stage builds
- use a `golang` image to build it
- copy the results to a another image

The result is a small Docker container built for Linux

And you can build it without even having Go installed!

This is great for CI/CD environments

Dockerfile extracts (1)

```
FROM golang:1.15-alpine AS builder

RUN /sbin/apk update && /sbin/apk --no-cache add ca-certificates \
git tzdata && /usr/sbin/update-ca-certificates

RUN adduser -D -g '' sort
WORKDIR /home/sort

COPY go.mod /home/sort
COPY go.sum /home/sort
COPY cmd    /home/sort/cmd
COPY *.go   /home/sort

ARG VERSION

RUN CGO_ENABLED=0 go build -a -tags netgo,osusergo \
    -ldflags "-extldflags '-static' -s -w" \
    -ldflags "-X main.version=$VERSION" -o sort ./cmd/sort
```

Dockerfile extracts (2)

```
FROM busybox:musl

COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/
COPY --from=builder /usr/share/zoneinfo /usr/share/zoneinfo
COPY --from=builder /etc/passwd /etc/passwd
COPY --from=builder /home/sort/sort /home/sort

USER sort
WORKDIR /home
EXPOSE 8081

ENTRYPOINT ["/home/sort"]
```

And build it:

```
docker build -t sort-anim:latest . -f build/Dockerfile --build-arg VERSION=$(VERSION)
docker tag  ${NAME}:latest ${HOST}/sort-anim:${VERSION}
docker push ${HOST}/sort-anim:${VERSION}
```