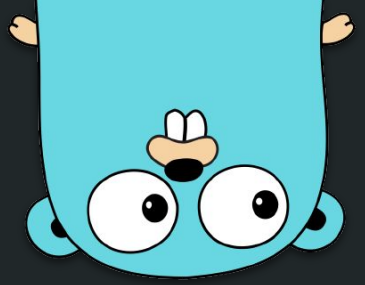
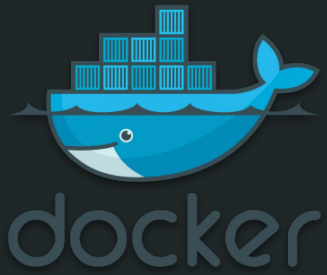




Docker for Go devs

Getting started + Tips

Artem Yadelskyi ● March 2023

Agenda

- Design Go applications with Docker in mind
  - Just use Go
  - The Twelve-Factor App (12factor.net)
- Dockerfile
  - Basics
  - Multi-stage build
  - Local run and debug with dlv and air
  - Choosing base images
  - Health check
- CI/CD
  - Run tests & linters (with/without Docker)
  - Build images & push to registry
  - Deploy
- Dev workflow
  - Docker compose
  - Makefile/Taskfile
  - Reading logs locally
  - Image cache & issues

Disclaimer

In this talk, all arguments and examples may be opinionated and may not follow best practices or some conventions. Generally this is my findings and that may differ from yours.

Design Go applications with Docker in mind

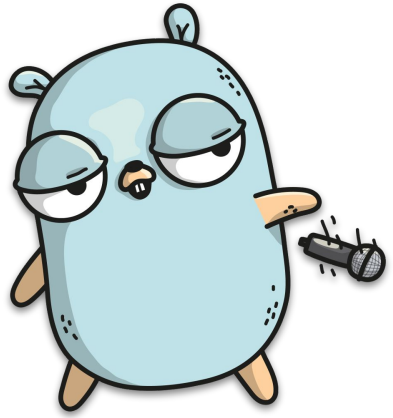
Steps:

...

Design Go applications with Docker in mind

Steps:

1. Just use Go!
2. End



Just use Go

Things that we just get out of the box:

- Single binary executable
- No external dependencies to run (if CGO is disabled)
- Cross-platform code
- Go modules (+ Go 1 promise of compatibility)
- Fast competition time
- Great tool ecosystem
- Relatively low CPU & RAM usage*
- ...

The Twelve-Factor App

12factor.net is a methodology for building software-as-a-service (SaaS) applications.

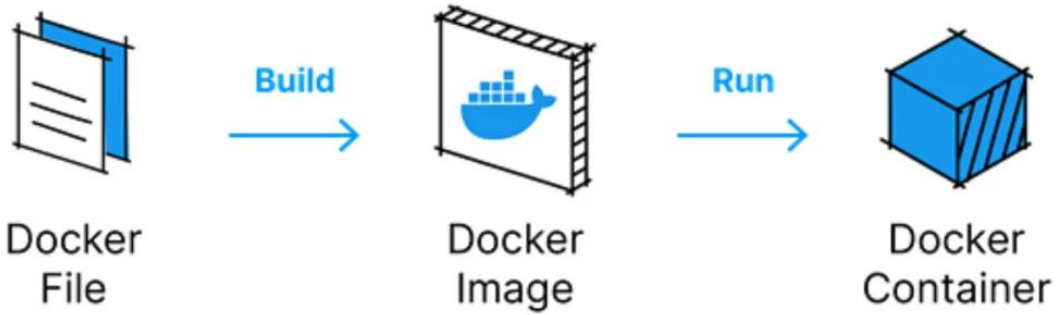
It provides a set of guidelines for creating scalable, maintainable, and portable applications that can run on any cloud platform.

The Twelve-Factor App

Most important for Docker:

- II. Dependencies (Explicitly declare and isolate dependencies)
- III. Config (Store config in the environment)
- VII. Port binding (Export services via port binding)
- VIII. Scalability/Concurrency (Scale out via the process model)
- IX. Disposability (Maximize robustness with fast startup and graceful shutdown)

Dockerfile



Dockerfile basics

```
FROM golang:1.20

WORKDIR /usr/src/app

# Pre-copy/Cache go.mod for pre-downloading dependencies and
# only redownloading them in subsequent builds if they change
COPY go.mod go.sum ./
RUN go mod download && go mod verify

COPY . .
RUN go build -v -o /usr/local/bin/app ./...

CMD ["app"]
```

Multi-stage build

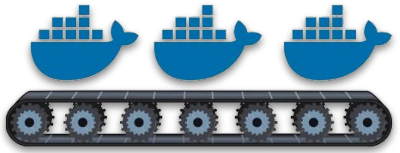
Split & repurpose different pieces of your Dockerfile

```
FROM golang:1.20 AS build

# ...

RUN go build -o app ./...

#  ==== ==== ==== ====  #

FROM ubuntu:bionic AS run

COPY --from=build app ./

CMD ["./app"]
```

Multi-stage build

Common stages that can be useful:

- Source (source code + dependencies)
- Dev (local development with hot reloading)
- Test (run tests + linters + etc.)
- Build (build executable)
- Release (final release image)



Multi-stage build

Source (source code + dependencies)

```
FROM golang:1.20-alpine AS source

WORKDIR /demo

RUN go env -w CGO_ENABLED="0"

COPY go.mod go.sum ./
RUN go mod download && go mod verify

COPY . .
```

Multi-stage build

Dev (local development with hot reloading)

```
FROM source AS dev

RUN go install github.com/go-delve/delve/cmd/dlv@latest && \
    go install github.com/cosmtrek/air@latest

ENTRYPOINT air
```

Multi-stage build

Test (run tests + linters + etc.)

```
FROM source AS test

RUN go test ./...
```

Multi-stage build

Build (build executable)

```
FROM source AS build

RUN apk --update add ca-certificates upx && update-ca-certificates

RUN go build -ldflags="-s -w" -o /bin/demo . && upx --best --lzma /bin/demo
```

go build	go build + ldflags	go build + ldflags + upx
16.6mb	11.9mb	3.5mb
0.96sec	0.93sec	7.25sec

From: Shrinking Go executables

Multi-stage build

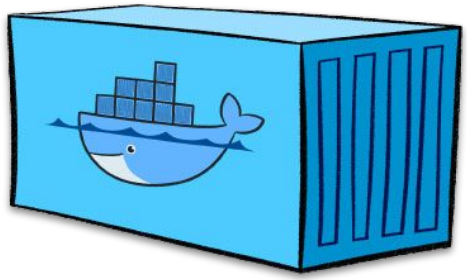
Release (final release image) + Health check

```
FROM scratch AS release

COPY --from=mymmrac/mini-health:latest /mini-health /mini-health
HEALTHCHECK CMD ["/mini-health", "/health"]

COPY --from=build /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/ca-certificates.crt
COPY --from=build /bin/demo /demo

ENTRYPOINT ["/demo"]
```

ENTRYPOINT: [“demo”] CMD: [“help”] Result: demo help

Base image

- golang:<version> (Debian) 301mb
- golang:<version>-alpine 100mb
- build image +
  - debian:<version> 53mb
  - debian:<version>-slim 30mb
  - ubuntu:<version> 26mb
  - alpine:<version> 3mb
  - scratch 0mb
  - ...



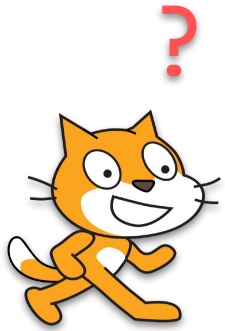
Scratch base image

Possible issues & limitations*:

- No shell
- No ca-certificates
- No debugger
- No CGO support
- No easy way to do health check
- ...

Benefits:

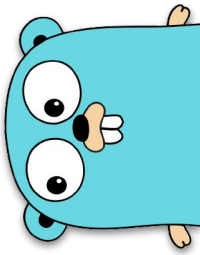
- Image size ⇒ deploy speed
- More secure*



Health check

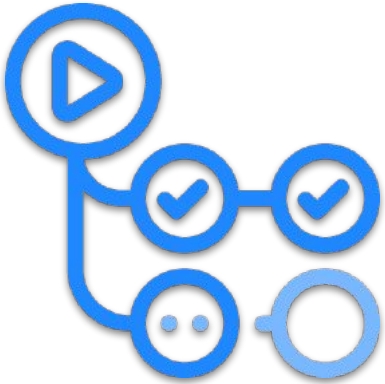
- Use fast & basic health check
- Depending on environment, health check might be redundant
- Handle panic and at least log them

CI/CD



Steps:

- Run tests/linters
  - In Docker image
  - Just as in regular CI
- Build
- Scan/test image
- Push to registry



Tests/linters in Docker

Benefits:

- No environment setup needed
- Run unit/performance/integration tests
- ...

Scan/test image

- Use tools like trivy, docker-scan-plugin, docker-bench-security, AWS ECR, etc. to scan images
- Check licenses
- Create containers in CI to test
- ...

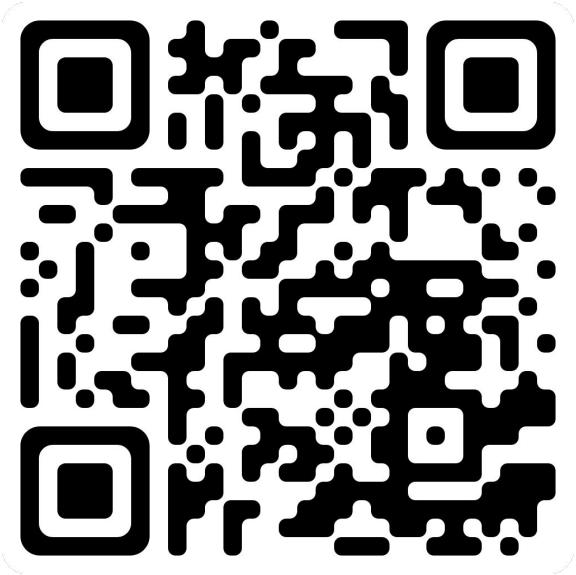
Dev workflow

Make your development easier with:

- Docker compose
- Makefile/Taskfile
- Logs from Docker locally
- Debug right app running in Docker
- ...

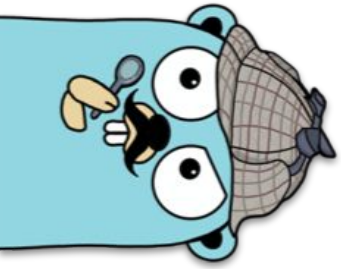
Cache everything, what can go wrong?

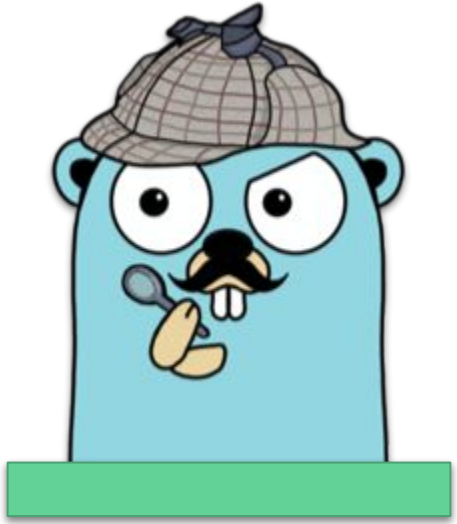
Demo



github.com/mymmrac/go-docker-demo

Q & A





Thanks for listening and keep Going!