# Advanced Techniques for Building Container Images

Adrian Mouat
@adrianmouat

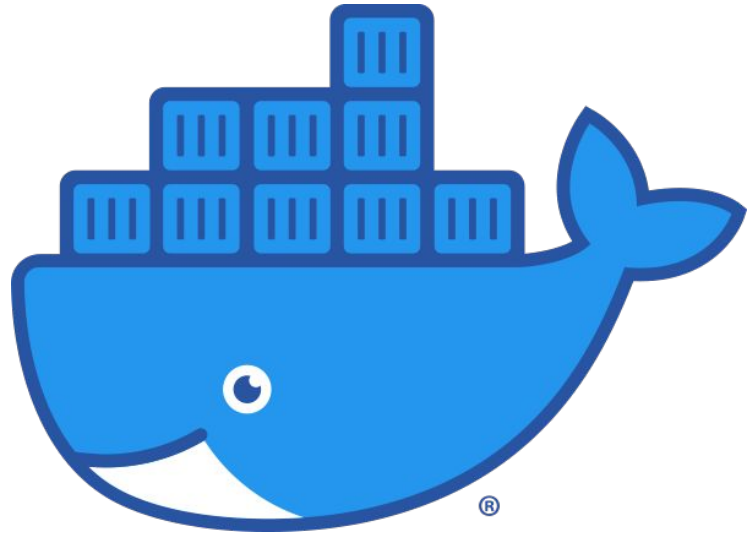
Container Solutions

info@container-solutions.com
www.container-solutions.com

Photo by Kevin Wood

# Docker

- *Build, Ship, Run*
- Build overshadowed by orchestration
- Recent focus on deployment not development



@adrianmouat

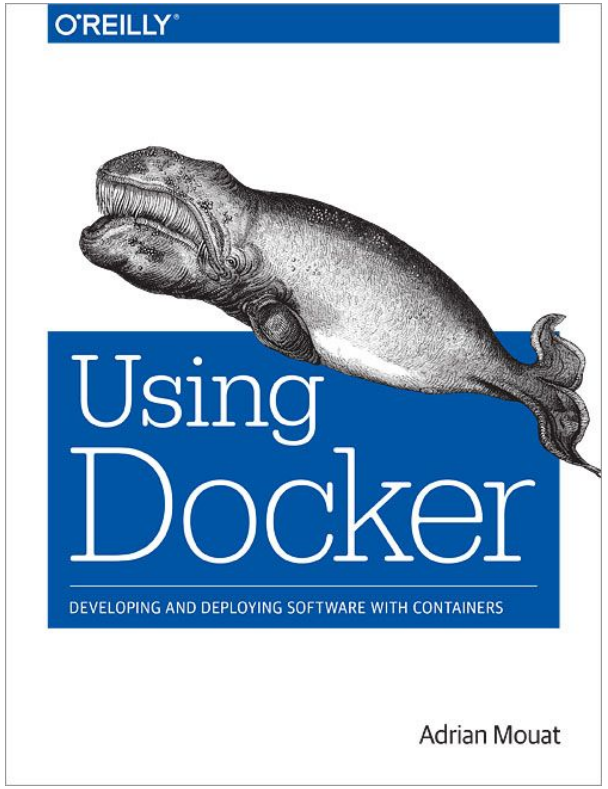

# Container Native Development

Using `docker build` instead of locally installed language tooling

@adrianmouat

# Python Dev with Docker

- Replace virtualenv
- Use volumes for live development
- Portable
- Reproducible



# Go Dev with Docker

- Still advantages
  - No installation
  - Consistent env
  - Easy for end users
- But a killer disadvantage...

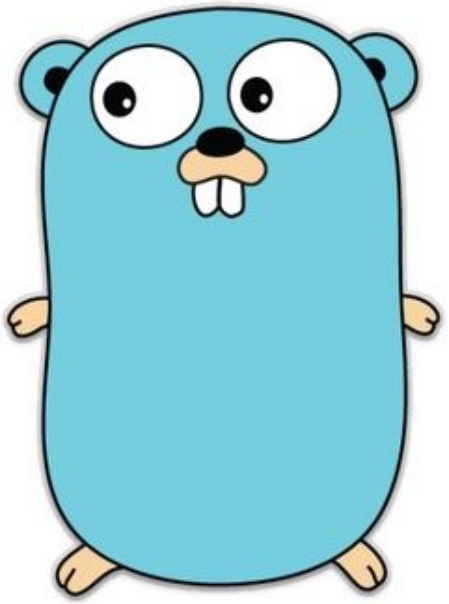


Photo by Jürgen Schoner

# Docker Build Problems

- Slower than local compilation
- Simplistic caching
- Requires root
- Secrets
- Dockerfile stopped evolving

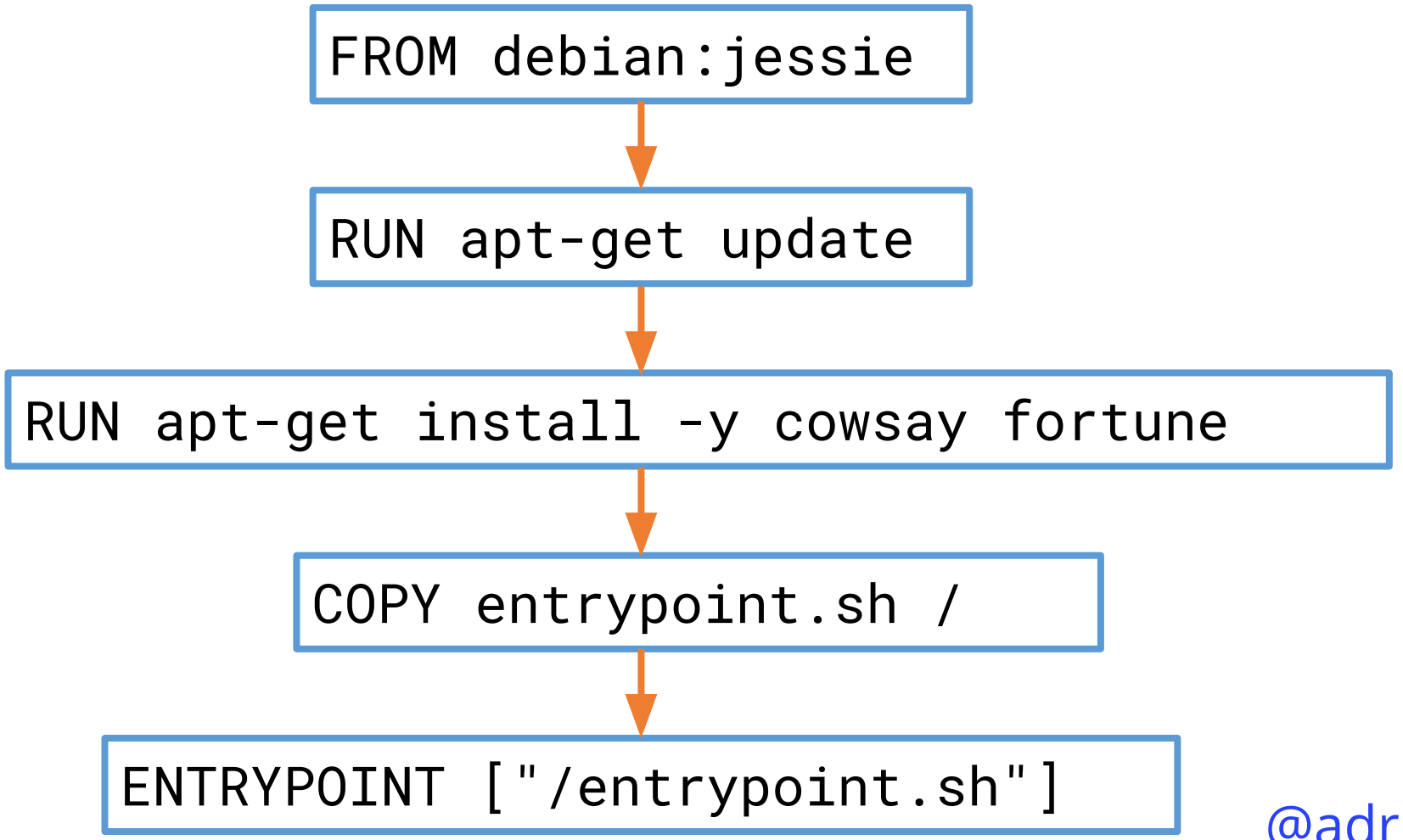
# Enter BuildKit

- Fundamental rewrite of backend
- Still client server model
  - (but see img by Jessie Frazelle)
- Intermediate representation
  - LLB

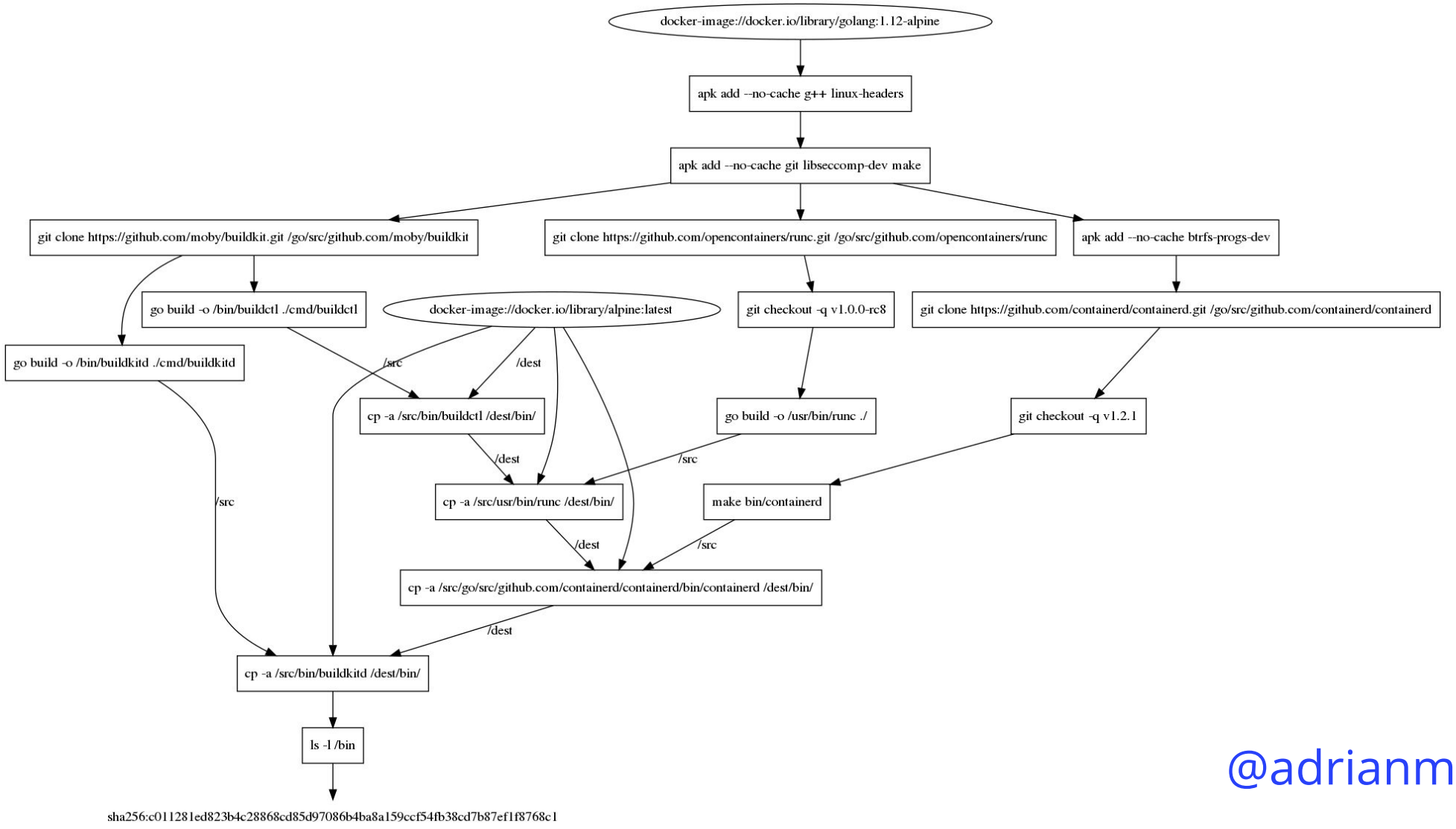
# Low Level Builder (LLB)

- Intermediate format for compiler
  - Dockerfiles etc are really code
- Similar idea to LLVM IR
  - Also Java bytecode, .NET CIL
- Forms a Graph (DAG)

# Simple Chain



# Graph




@adrianmouat

# Frontends

- Source that compiles to LLB
- New dockerfile frontend
- Handful of others
- Essential to exploit parallelism

# Other Highlights

- New mount options
- Output formats
- Distributable workers
- Cross-compilation
- Rootless execution
- bake

# Dev Speed Contest

- `go build` vs `docker build`
- Runc project


64
OPF 4E
17
WOO II

# Close Enough?

- Maybe
- Definitely for occasional bug fixes
- Maybe not for full-time

# IDEs

- Incremental compilation
- Could IDEs use containers?
  - Team gets same settings and libs

# Other New Stuff

# Mount Options

- We've seen cache. Also
  - `bind`
    - Volume from build context, read-only
  - `tmpfs`
    - In-memory

# Mount Options

- `secret` and `ssh`
- Allow sensitive date to be used but not leaked in final image
- Requires build arguments as well Dockerfile changes

# secret Example

```
# syntax = docker/dockerfile:experimental
FROM python:3
RUN pip install awscli
RUN --mount=type=secret,id=aws,target=/root/.aws/credentials aws
s3 cp s3://... ...
```

- `docker build --secret id=aws,src=$HOME/.aws/credentials \`

  `-t my/image .`

@adrianmouat

# More cache

- `--cache-from`
  - Load cache from existing image!

https://asciinema.org/a/bZrOoCDK6oChNYfYD8QlY8crB

# buildx

- Buildx is an experimental plugin for Docker
- Effectively separate binary
  - Standalone buildkit
- Can talk to multiple builder instances
- Instances can be Docker or buildkit
- Also some new commands

# buildx

```
$ docker buildx --help
Usage: docker buildx COMMAND
Build with BuildKit
Management Commands:
  imagetools  Commands to work on images in registry
Commands:
  bake        Build from a file
  build       Start a build
  create      Create a new builder instance
  inspect     Inspect current builder instance
  ls          List builder instances
  rm          Remove a builder instance
  stop        Stop builder instance
  use         Set the current builder instance
  version     Show buildx version information
```

@adrianmouat

# buildx

```
$ docker buildx ls
NAME/NODE       DRIVER/ENDPOINT             STATUS   PLATFORMS
second-build    docker-container
  second-build0 unix:///var/run/docker.sock inactive
default *       docker
  default       default                     running  linux/amd64,
linux/arm64, linux/ppc64le, linux/s390x, linux/386, linux/arm/v7,
linux/arm/v6
```

...

@adrianmouat

# So distributed builds?

- Not quite
- Instances support different platforms
  - different builders for arm etc
- But not one build across multiple instances :(

# Multiplatform Builds

- `docker build --platform=linux/arm64 .`
- Works, assuming your base images support arm64 etc
- By default, uses QEMU under the hood
- Can list multiple platforms
- Or use buildx instances for given platforms...
- Can also use language tooling!

# Concurrent Builds

- Exploit parallelism in LLB DAG
- With Dockerfile, can use multistage builds
- True parallelism requires more intelligent front ends

# Bake

- Personally, I **hate** Make
- But shell scripts and Makefiles are common with Docker
- Calling `docker build` from Make usually sequential

# Enter Bake

```
group “default” {
   targets = [“db”, “webapp-dev”] }

target “webapp-dev” {
   dockerfile = "Dockerfile.webapp"
   tags = ["docker.io/username/webapp"] }

target “webapp-release” {
   inherits = [“webapp-dev”]
   platforms = [“linux/amd64”, “linux/arm64”]
}
```

@adrianmouat

# Bake

- `docker buildx bake`
- `docker buildx bake release`
- `docker buildx bake test validate lint`
- `docker buildx bake binaries-cross`
- `docker buildx bake help`

# Conclusion

- Way we deploy and run software has changed
  - Microservices
  - Kubernetes
  - Service Mesh
  - Observability

# Conclusion

- Way we develop and build has changed
  - But further to go
  - Container Native?
    - LLB frontends, IDE integration
  - Cluster Native?
    - Tilt, Skaffold, Draft
    - Darklang

# References

- DockerCon presentation on buildkit internals and frontends by Tonis Tiigi and Matt Rickard
  - https://www.youtube.com/watch?v=x5zDN9_c-k4
  - https://docs.google.com/presentation/d/1maienHIl8FtCmTcx8QFb_ieM9ElDoOY1HrX8YnsxvRQ/
- Mockerfile blog https://matt-rickard.com/building-a-new-dockerfile-frontend/
- Buildkit https://github.com/moby/buildkit/
- Solver design https://github.com/moby/buildkit/blob/master/docs/solver.md

# New Docker Build Stuff

- Turn on with `export DOCKER_BUILDKIT=1`
  - Assuming using 19.03

# Old Style Output

```
$ docker build --no-cache -f Dockerfile.debug .
Sending build context to Docker daemon  10.46MB
Step 1/30 : FROM rust:latest as builder
 ---> 385005cad312
Step 2/30 : RUN rustup update nightly && rustup default
nightly;
 ---> Running in b7ae81349a22
...
```

# New Style Output

```
File  Edit  View  Search  Terminal  Help
[+] Building 547.1s (30/32)
 => [internal] load build context                                                0.3s
 => => transferring context: 431.69kB                                            0.2s
 => CACHED [stage-1 1/7] FROM docker.io/library/debian:stable-slim               0.0s
 => [builder 2/20] RUN rustup update nightly && rustup default nightly;         47.4s
 => [stage-1 2/7] RUN apt-get update  && apt-get install -y --no-install-recommends openssl libssl-dev  18.6s
 => [builder 3/20] RUN apt-get update && apt-get install -y cmake golang unzip  26.1s
 => [builder 4/20] RUN cd /usr/local && curl -o protoc.zip -sSL https://github.com/google/protobuf/rel  19.0s
 => [builder 5/20] WORKDIR /usr/src/trow                                        13.6s
 => [builder 6/20] COPY Cargo.toml .                                            15.3s
 => [builder 7/20] COPY Cargo.lock .                                            14.3s
 => [builder 8/20] RUN mkdir src/                                               14.0s
 => [builder 9/20] RUN echo "fn main() {}" > src/main.rs                        17.6s
 => [builder 10/20] COPY lib/server/Cargo.toml lib/server/                      11.9s
 => [builder 11/20] RUN mkdir -p lib/server/src                                 14.5s
 => [builder 12/20] RUN touch lib/server/src/lib.rs                             13.3s
 => [builder 13/20] COPY lib/protobuf/Cargo.toml lib/protobuf/                  12.8s
 => [builder 14/20] RUN mkdir -p lib/protobuf/src                               15.0s
 => [builder 15/20] RUN touch lib/protobuf/src/lib.rs                           18.3s
 => [builder 16/20] RUN cargo fetch #This should be cargo build, but it fails as we need build libs fi  64.6s
 => [builder 17/20] COPY lib lib                                                15.4s
 => [builder 18/20] COPY src src                                                19.6s
 => [builder 19/20] RUN touch src/main.rs                                       25.4s
 => [builder 20/20] RUN cargo build                                            159.5s
 => [stage-1 3/7] COPY --from=builder /usr/src/trow/target/debug/trow /trow      4.4s
 => [stage-1 4/7] COPY install/self-cert /install/self-cert                      0.5s
 => [stage-1 5/7] COPY start-trow.sh /                                           0.4s
 => [stage-1 6/7] RUN mkdir --parents /data/layers                               3.1s
```

# Custom Outputs

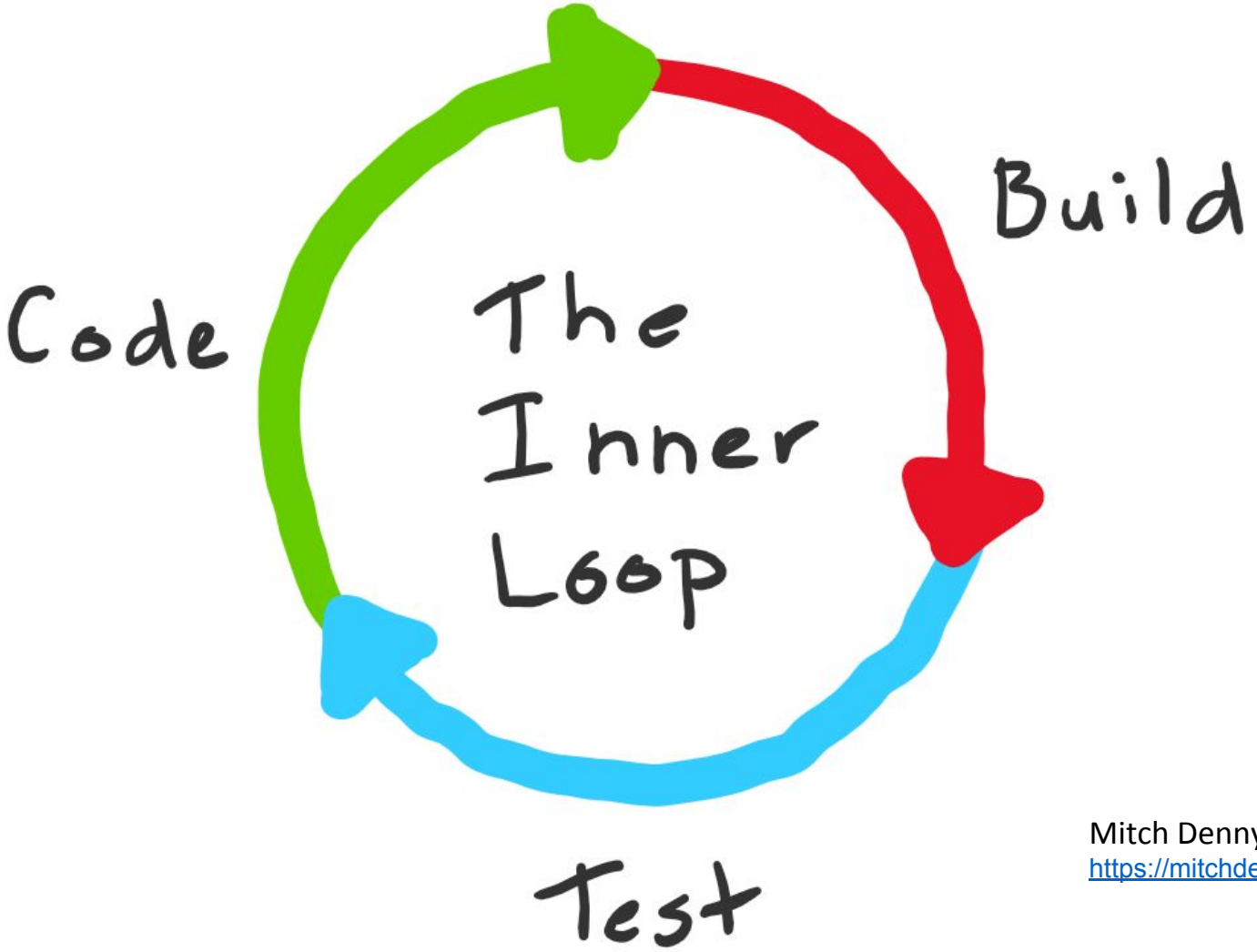
- What if you want a binary or other artifact e.g. pdf?
- Traditionally have to use `docker cp`
- Now we can do something like:

```
docker build --output . .
```

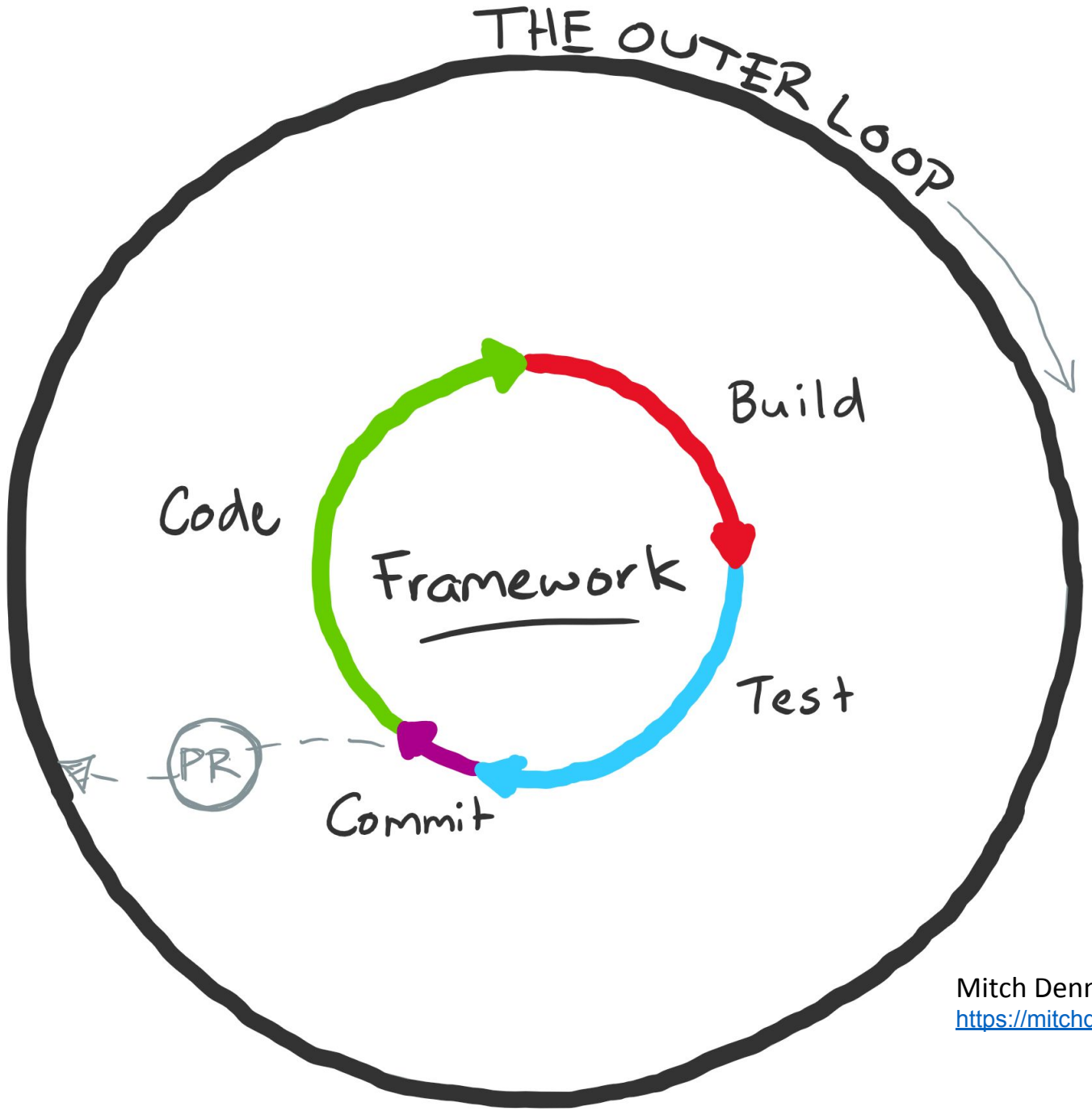
# Mockerfile Example

- https://matt-rickard.com/building-a-new-dockerfile-frontend/
- https://github.com/r2d4/mockerfile/blob/master/Mockerfile.yaml

# Development

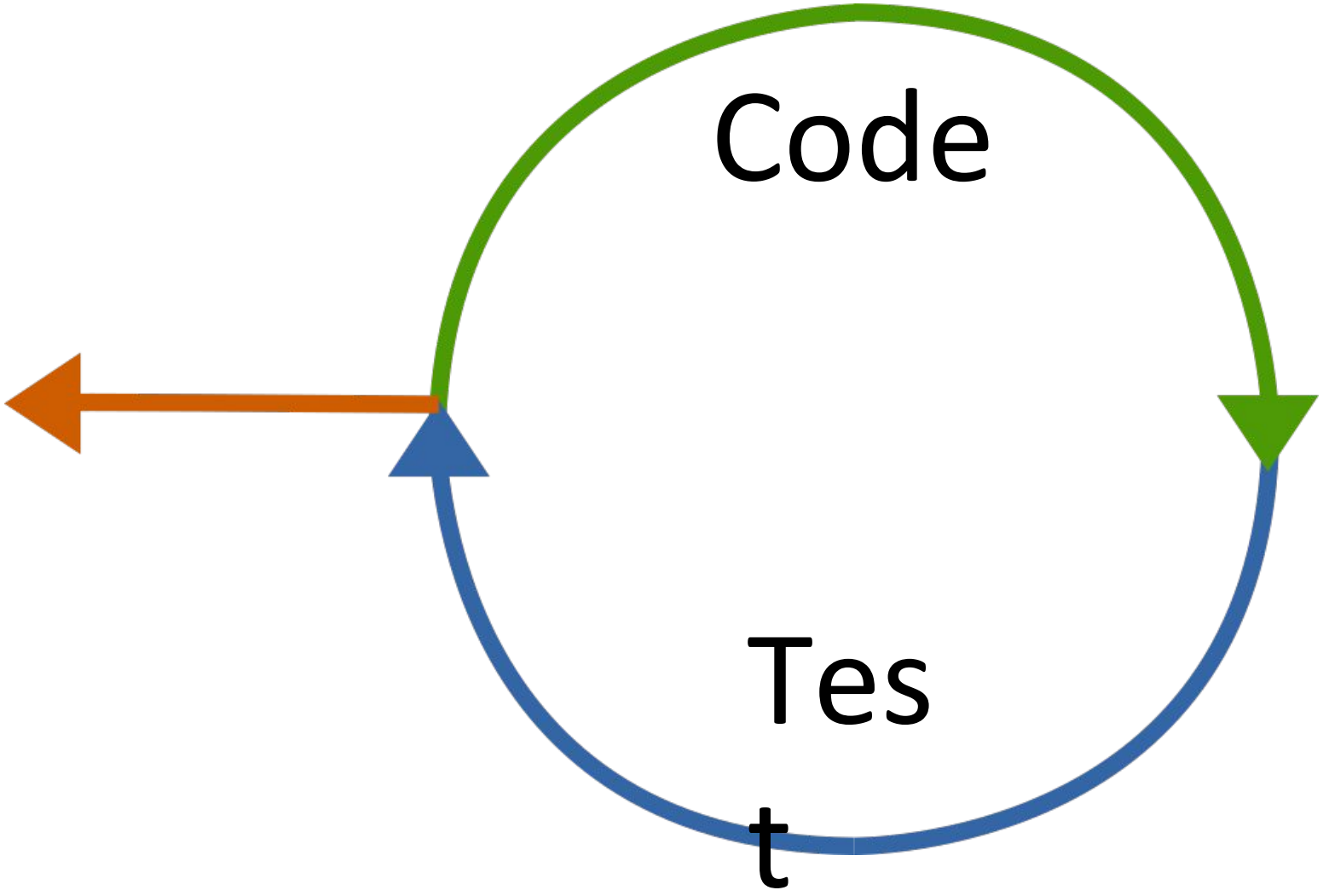


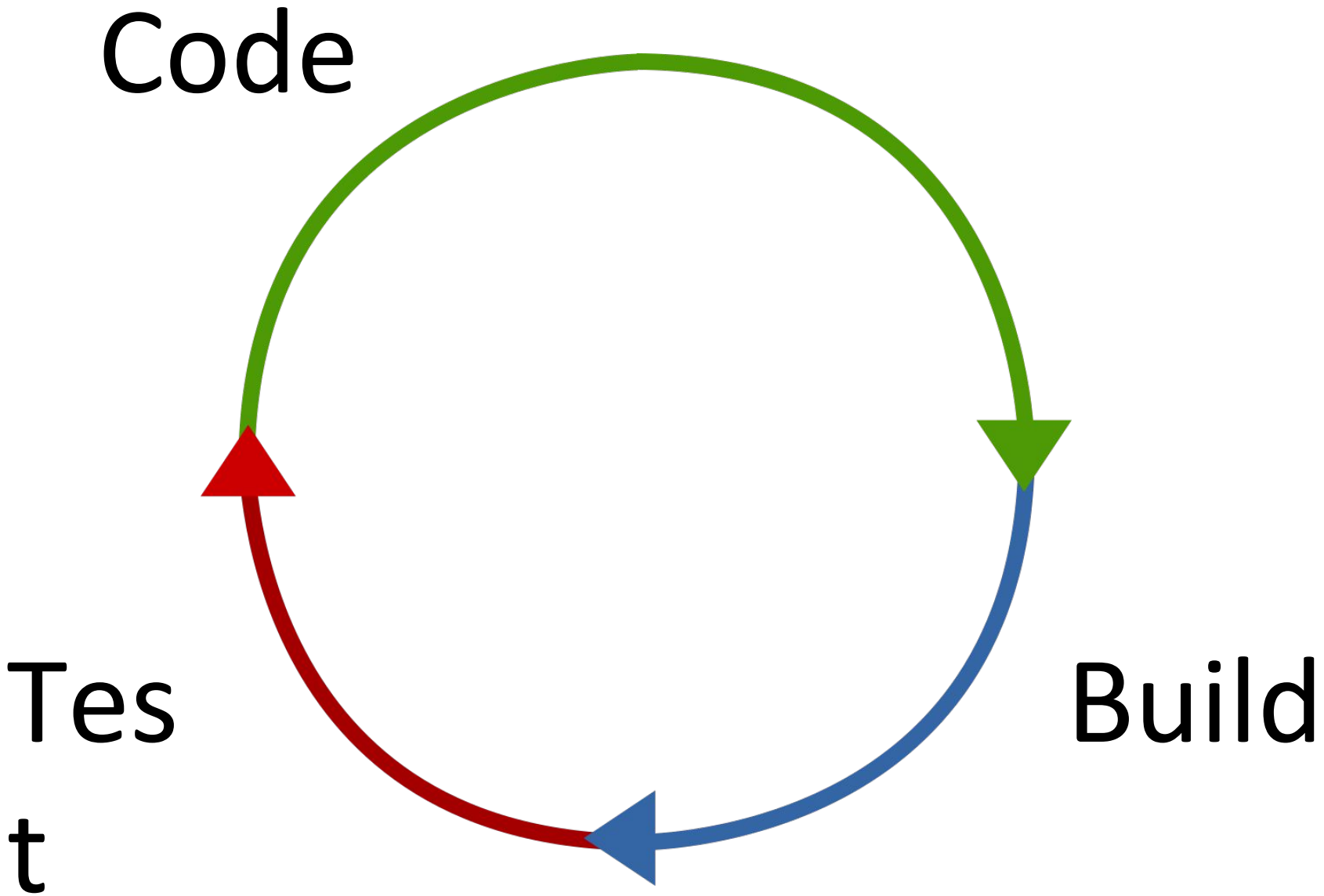

Mitch Denny The Inner Loop
https://mitchdenny.com/the-inner-loop/




Mitch Denny The Inner Loop
https://mitchdenny.com/the-inner-loop/

# Build?




Code
Build
Tes
t

# Can Docker be in the Loop?