ko Go images, no Dockerfile

CONCEPTS

ko runs go build on your machine and layers the static binary onto a small base image: no Dockerfile and no Docker daemon. It suits one pure-Go binary per image.

Go package package main
↓ `go build`
ko layer on base image
↓ `push`
KO_DOCKER_REPO digest reference
↓ `ko://` replaced
Kubernetes YAML resolve / apply

ko compiles locally and pushes straight to the registry; YAML is rewritten on the way to the cluster

Image names under KO_DOCKER_REPO

Flag	Resulting name
`(default)`	<repo>/<pkg>-<md5 of import path>
`-B`	<repo>/<last path element>
`-P`	<repo>/<full import path>
`--bare`	<repo> itself

Special KO_DOCKER_REPO values: ko.local loads into the local Docker daemon, kind.local loads into a kind cluster.

BUILD & PUBLISH

`ko build <importpath>`
Build, push, print image digest

`ko build -B <importpath>`
Short name, no hash suffix

`ko build -t <tag>`
Tag instead of only latest

`ko build --push=false <importpath>`
Build without pushing

`ko build --local <importpath>`
Load into local Docker daemon

`ko build --bare <importpath>`
Use KO_DOCKER_REPO as the full name

`ko build --image-refs <file> <importpath>`
Write published refs to a file

`ko build --ldflags <flags> <importpath>`
Pass ldflags to go build

PLATFORMS & IMAGES

`ko build --platform=all <importpath>`
Every platform the base supports

`ko build --platform=linux/amd64,linux/arm64 <importpath>`
Chosen platforms, one index

`ko build --image-label <k>=<v> <importpath>`
Add an image label

`ko build --sbom=none <importpath>`
Skip SBOM generation

Platform value forms

Value	Meaning
`all`	All platforms of the base image
`linux/arm64`	os/arch
`linux/arm/v7`	os/arch/variant
`a,b`	Comma-separated list

KUBERNETES

`ko resolve -f <path>`
Print YAML with images built and pinned

`ko resolve -f <path> | kubectl apply -f -`
Resolve and apply by hand

`ko apply -f <path>`
Resolve and kubectl apply

`ko apply -f <path> -- <kubectl flags>`
Pass flags through to kubectl

`ko apply -R -f <dir>`
Recurse into subdirectories

`ko resolve -l <key>=<value> -f <path>`
Only objects matching a label

`ko create -f <path>`
Resolve and kubectl create

`ko delete -f <path>`
kubectl delete; builds nothing

`ko run <importpath>`
Build and run as a one-off pod

`ko run <importpath> -- <app args>`
Pass arguments to the app

ko:// reference in a manifest

```
containers:
- name: app
  image: ko://github.com/<user>/
<repo>/cmd/app
```

CONFIG & ENVIRONMENT

KO_DOCKER_REPO
Registry to push to (required)

KO_DEFAULTBASEIMAGE
Override the default base image

KO_DEFAULTPLATFORMS
Default platforms, comma-separated

KO_CONFIG_PATH
Path to the config file

KO_DATA_PATH
In-container path of kodata files

.ko.yaml top-level keys

Key	Sets
`defaultBaseImage`	Base image for all builds
`baseImageOverrides`	Base image per import path
`defaultPlatforms`	Target platforms
`defaultEnv / defaultFlags / defaultLdflags`	Build defaults
`builds`	Per-build id, dir, main, env, flags, ldflags

Add a kodata/ directory next to the main package to ship static files. Read KO_DATA_PATH in code; set it yourself with go run to test locally. Symlinks in kodata are followed.

AUTH

`ko login <registry> -u <user> -p <pass>`
Log in to a registry

`ko login <registry> -u <user> --password-stdin`
Read the password from stdin

`ko version`
Print the ko version

ko reuses your Docker config: if docker push works, ko works. It also has built-in credential helpers for GCR, ECR, ACR and GHCR (GITHUB_TOKEN).