ko Go images, no Dockerfile

START HERE

`export KO_DOCKER_REPO=` *`<registry/repo>`*
Where images are pushed

`ko apply -f config/`
Resolve, then kubectl apply

`ko build ./cmd/`*`<app>`*
Build, push, print digest

`ko build --local ./cmd/` *`<app>`*
Load into local Docker

`ko run ./cmd/`*`<app>`*
Build and run in the cluster

`ko build --platform=all ./cmd/`*`<app>`*
Multi-arch image

`ko resolve -f config/`
YAML with images filled in

`ko login` *`<registry>`* **`-u`** *`<user>`*
Store registry credentials

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

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.

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
The package must be package main with a func main. The binary lands at /ko-app/<name> and is the entrypoint.

`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 -P` *`<importpath>`*
Keep full import path in the name

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

`ko build --tag-only` *`<importpath>`*
Refs carry tags, not digests

`ko build --tarball` *`<file>`* *`<importpath>`*
Save image as a tarball

`ko build --oci-layout-path` *`<dir> <importpath>`*
Save as an OCI layout

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

BUILD & PUBLISH (cont.)

`ko build --insecure-registry` *`<importpath>`*
Skip TLS verification

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 --image-annotation` *`<k>=<v>`* *`<importpath>`*
Add an OCI manifest annotation

`ko build --image-user` *`<user> <importpath>`*
User the image runs as

`ko build --sbom=none` *`<importpath>`*
Skip SBOM generation
An SPDX SBOM is built and uploaded by default.

`ko build --sbom-dir` *`<dir>`* *`<importpath>`*
Also write the SBOM to disk

`ko build --debug` *`<importpath>`*
Wrap app in Delve on port 40000

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
Output goes to stdout. Each ko://<importpath> value is replaced by a digest reference.

`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

KUBERNETES (cont.)

`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
Does not remove images already pushed.

`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
Default is cgr.dev/chainguard/static.

KO_DEFAULTPLATFORMS
Default platforms, comma-separated

KO_CONFIG_PATH
Path to the config file

KOCACHE
Cache dir; skips unchanged builds

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

CONFIG & ENVIRONMENT (cont.)

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.

ldflags and flags in .ko.yaml accept templates such as {{.Env.VERSION}}, {{.Git.ShortCommit}} and {{.Date}}.

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).

RECIPES

Build, push, deploy

```
export KO_DOCKER_REPO=ghcr.io/
<user>/<repo>
ko apply -f config/
# manifests use image: ko://
github.com/<user>/<repo>/cmd/
app
```

Multi-arch release with a version tag

```
export KO_DOCKER_REPO=ghcr.io/
<user>/<repo>
ko build -B -t v1.2.3 --
platform=linux/amd64,linux/
arm64 ./cmd/app
```

Try an image locally

```
ko build --local -B ./cmd/app
docker run --rm ko.local/
app:latest
```

Render a release manifest

```
ko resolve -f config/ >
release.yaml
kubectl apply -f release.yaml
```