Bake targets

- Default target
- Target properties
- Grouping targets
  - Run tests and lint checks
- Export build artifacts
- Pattern matching for targets and groups
- Additional resources

A target in a Bake file represents a build invocation. It holds all the information you would normally pass to a `docker build` command using flags.

```
target "webapp" {
  dockerfile = "webapp.Dockerfile"
  tags = ["docker.io/username/webapp:latest"]
  context = "https://github.com/username/webapp"
}
```

To build a target with Bake, pass name of the target to the `bake` command.

```
$ docker buildx bake webapp
```

You can build multiple targets at once by passing multiple target names to the `bake` command.

```
$ docker buildx bake webapp api tests
```

Default target

If you don't specify a target when running `docker buildx bake`, Bake will build the target named `default`.

```
target "default" {
  dockerfile = "webapp.Dockerfile"
  tags = ["docker.io/username/webapp:latest"]
  context = "https://github.com/username/webapp"
}
```

To build this target, run `docker buildx bake` without any arguments:

```
$ docker buildx bake
```

Target properties

The properties you can set for a target closely resemble the CLI flags for `docker build`, with a few additional properties that are specific to Bake.

The `dockerfile` property specifies the path to the Dockerfile for a target. If you also set a `context`, the `dockerfile` path resolves relative to that context.

```
target "default" {
  context = "app"
  # resolves to app/src/www/Dockerfile
  dockerfile = "src/www/Dockerfile"
}
```

For all the properties you can set for a target, see the Bake reference.

Grouping targets

You can group targets together using the `group` block. This is useful when you want to build multiple targets at once.

```
group "all" {
  targets = ["webapp", "api", "tests"]
}

target "webapp" {
  dockerfile = "webapp.Dockerfile"
  tags = ["docker.io/username/webapp:latest"]
```

```
  context = "https://github.com/username/webapp"
}

target "api" {
  dockerfile = "api.Dockerfile"
  tags = ["docker.io/username/api:latest"]
  context = "https://github.com/username/api"
}

target "tests" {
  dockerfile = "tests.Dockerfile"
  contexts = {
    webapp = "target:webapp"
    api = "target:api"
  }
  output = ["type=local,dest=build/tests"]
  context = "."
}
```

To build all the targets in a group, pass the name of the group to the `bake` command.

```
$ docker buildx bake all
```

Run tests and lint checks

Bake targets can run checks without producing an image. For a Go project with a `go.mod` file, source code, and tests, define matching Dockerfile stages:

```
FROM --platform=$BUILDPLATFORM golang:{{% param "example_go_version" %}} AS base
WORKDIR /src
COPY . .

FROM base AS test
RUN go test ./...

FROM base AS lint
RUN go vet ./...
```

Map each Bake target to its Dockerfile stage and group the checks:

```
group "validate" {
  targets = ["test", "lint"]
}

target "test" {
  target = "test"
  output = ["type=cacheonly"]
}

target "lint" {
  target = "lint"
  output = ["type=cacheonly"]
}
```

```
$ docker buildx bake validate
```

Both checks run concurrently. A failing check causes the command to fail. The `cacheonly` output keeps the build cache without exporting an image.

Export build artifacts

A target can export files, such as compiled binaries, to your local filesystem. For the Go project in the previous example, append these stages to the Dockerfile:

```
FROM base AS build
ARG TARGETOS
ARG TARGETARCH
ARG BUILD_MODE=release
RUN if [ "$BUILD_MODE" = "debug" ]; then \
      CGO_ENABLED=0 GOOS=$TARGETOS GOARCH=$TARGETARCH go build -gcflags="all=-N -l" -o
/out/app .; \
    else \
      CGO_ENABLED=0 GOOS=$TARGETOS GOARCH=$TARGETARCH go build -o /out/app .; \
    fi

FROM scratch AS export
COPY --from=build /out/ /
```

The `scratch` stage contains only the files to export. Add these Bake targets to save the binary locally or build it for multiple platforms:

```
target "bin" {
  target = "export"
  platforms = ["local"]
  output = ["type=local,dest=build/bin"]
}

target "bin-cross" {
  inherits = ["bin"]
  platforms = ["linux/amd64", "linux/arm64"]
}
```

```
$ docker buildx bake bin
$ docker buildx bake bin-cross
```

The `bin` target builds for the client's platform and writes `build/bin/app`. The `bin-cross` target writes separate `build/bin/linux_amd64/app` and `build/bin/linux_arm64/app` files. See Inheritance and Local and tar exporters for reuse and output options.

The base stage uses `BUILDPLATFORM` to run the Go compiler on the builder's native platform. `TARGETOS` and `TARGETARCH` select the output platform, so this example cross-compiles without emulation. For other toolchains, see Multi-platform build strategies.

Pattern matching for targets and groups

Bake supports shell-style wildcard patterns when specifying target or grouped targets. This makes it easier to build multiple targets without listing each one explicitly.

Supported patterns:

- `*` matches any sequence of characters
- `?` matches any single character
- `[abc]` matches any character in brackets

> [!NOTE]
>
> Always wrap wildcard patterns in quotes. Without quotes, your shell will expand the wildcard to match files in the current directory, causing errors.

Examples:

```
# Match all targets starting with 'foo-'
$ docker buildx bake "foo-*"

# Match all targets
$ docker buildx bake "*"

# Matches: foo-baz, foo-caz, foo-daz, etc.
$ docker buildx bake "foo-?az"

# Matches: foo-bar, boo-bar
$ docker buildx bake "[fb]oo-bar"

# Matches: mtx-a-b-d, mtx-a-b-e, mtx-a-b-f
$ docker buildx bake "mtx-a-b-*"
```

You can also combine multiple patterns:

```
$ docker buildx bake "foo*" "tests"
```

Additional resources

Refer to the following pages to learn more about Bake's features:

- Learn how to use variables in Bake to make your build configuration more flexible.
- Learn how you can use matrices to build multiple images with different configurations in Matrices.
- Head to the Bake file reference to learn about all the properties you can set in a Bake file, and its syntax.

□□□ #2
□ □□□ □□□ 2026-10-31 19:03:36 UTC
□ □□□ □□□ 2026-10-31 19:03:38 UTC