Dagger Programmable CI pipelines

START HERE

`dagger init --sdk=go`
New module

`dagger call build terminal`
Shell into result

`dagger functions`
List functions

`dagger install <module>`
Add a dependency

`dagger call <fn>`
Run a function

`dagger develop`
Regenerate bindings

`dagger call <fn> --help`
Show its arguments

`dagger`
Interactive shell

Your code module + SDK → `dagger call` → **CLI** GraphQL API → `session` → **Engine** BuildKit-based → `runs` → **Containers** cached steps

CONCEPTS

Functions run in containers on the Dagger engine, so a pipeline behaves the same on your laptop and in CI.

Chain functions on the command line: each returned object exposes its own functions, such as container, directory and file.

Results are cached by content. A step reruns only when its inputs change.

MODULES

`dagger init --sdk=<go|python|typescript>`
Create a module in this directory
Writes dagger.json. Without --sdk, run dagger develop --sdk later.

`dagger init --name <name>`
Name the module

`dagger develop`
Regenerate SDK code and bindings
Run after installing a dependency or upgrading the engine.

`dagger install <module>`
Add a module as dependency
Example: github.com/shykes/daggerverse/hello@v0.3.0

`dagger install -n <name> <module>`
Install under another name

`dagger update <module>`
Update a dependency

`dagger uninstall <module>`
Remove a dependency

`dagger toolchain install <module>`
Add a toolchain to the module

`dagger toolchain list`
List installed toolchains

`dagger init --blueprint=<module>`
Start from a blueprint module

`dagger develop --recursive`
Develop local dependencies too

`dagger config`
View or edit module config

CALL FUNCTIONS

`dagger functions`
List the module's functions

`dagger functions <fn>`
List a function's sub-functions

`dagger call <fn>`
Run a function

`dagger call <fn> --help`
Show a function's arguments

`dagger call <fn> --<arg> <value>`
Pass an argument
Argument names are kebab-case versions of the code parameter names.

`dagger call <fn> <next-fn>`
Chain functions

`dagger call <fn> --source=.`
Pass a local directory
A Directory argument accepts a local path or a git URL.

`dagger call <fn> stdout`
Print a container's output

`dagger call <fn> -o <path>`
Save the result to a file or directory

`dagger call <fn> --json`
Print result as JSON

`dagger call <fn> terminal`
Open a shell in the result container

`dagger call <fn> export --path <path>`
Export a File or Directory

`dagger call -m <ref> <fn>`
Call a module by path or git ref
Example: -m github.com/shykes/daggerverse/hello

`dagger call -M <fn>`
Do not load the current module

SHELL AND API

`dagger`
Open the interactive Dagger shell
Pipe-like syntax, such as container | from alpine | terminal.

`dagger -c "<script>"`
Run a Dagger shell script

`dagger <file.dsh>`
Run a Dagger shell file

SHELL AND API (cont.)

`dagger core container from <image>`
Call the core API directly

`dagger core --help`
List core API objects

`dagger query < <file.graphql>`
Send a GraphQL query

`dagger query --var <k>=<v>`
Pass query variables

`dagger run <command>`
Run any program inside a Dagger session
Sets DAGGER_SESSION_PORT and DAGGER_SESSION_TOKEN for the program.

Shell, call or run?

Command	Use for
`dagger call`	functions in a module
`dagger (shell)`	exploring, ad-hoc pipelines
`dagger run`	your own SDK script

SECRETS

`dagger call <fn> --token=env://<VAR>`
Secret from an env variable
The value never appears in logs or the cache key.

`dagger call <fn> --token=file://<path>`
Secret from a file

`dagger call <fn> --token=cmd://"<command>"`
Secret from a command output

`dagger call <fn> --token=op://<vault>/<item>/<field>`
Secret from 1Password

Use the Secret type for tokens and keys, never a plain string argument, so values stay out of logs and traces.

OUTPUT AND DEBUG

`dagger call -i <fn>`
Shell into a failing exec
Opens a terminal at the step that failed.

`dagger call -v <fn>`
More detail (-vv, -vvv for more)

OUTPUT AND DEBUG (cont.)

`dagger call -q <fn>`
Less noise, keep progress

`dagger call -s <fn>`
Hide progress entirely

`dagger call --progress=plain <fn>`
Plain log output for CI
Others: auto, tty, dots, logs.

`dagger call -w <fn>`
Open the trace in a browser

`dagger call -E <fn>`
Keep the TUI open after finish

`dagger call -d <fn>`
Debug logs and full verbosity

`dagger call -y <fn>`
Auto-apply returned changesets
Changesets modify your local files, so review them first.

`dagger login`
Log in to Dagger Cloud

`dagger logout`
Log out of Dagger Cloud

RECIPES

Create and run a Go module

```
dagger init --sdk=go --name=ci
dagger functions
dagger call container-echo --
string-arg hi stdout
```

Test and build in CI

```
dagger call --progress=plain
test --source=.
dagger call build --source=.
export --path=./bin/app
```

Build a container and publish

```
dagger call build --source=. \
  publish --address=ghcr.io/
<org>/<app>:latest
```

Pass a token from the environment

```
export GH_TOKEN=<token>
dagger call deploy --
token=env://GH_TOKEN
```

HELP

`dagger --help`
List all commands and flags

`dagger <command> --help`
Help for one command

`dagger version`
Print the CLI version