Dagger Programmable CI pipelines

CONCEPTS

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

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

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

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

dagger develop
Regenerate SDK code and bindings

dagger install <*module*>
Add a module as dependency

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

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

dagger call <*fn*> **--source=.**
Pass a local directory

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

SHELL AND API

dagger
Open the interactive Dagger shell

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

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

dagger core --help
List core API objects

dagger run <*command*>
Run any program inside a Dagger session

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

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

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

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

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

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

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

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