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.

MODULES

`dagger init --sdk=<go|python|typescript>`
Create a module in this directory

`dagger develop`
Regenerate SDK code and bindings

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

`dagger update <module>`
Update a dependency

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

CALL FUNCTIONS (cont.)

`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> terminal`
Open a shell in the result container

`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 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

OUTPUT AND DEBUG

`dagger call -i <fn>`
Shell into a failing exec

`dagger call --progress=plain <fn>`
Plain log output for CI