Installing Drydock

Drydock is a Python command-line application. Installing it places the `drydock` executable and its versioned application resources into a Python environment. It does not create a project, a Blueprint, or a workspace in your home directory — you do that in a few explicit steps below.

The PyPI distribution is named `drydock-sdd`; the installed command is named `drydock`.

1. Prerequisites

Requirement	Why
Python 3.11 or later	Drydock's runtime floor.
`uv` or `pipx` (recommended)	Installs `drydock` as an isolated command-line tool.
A subscription provider CLI — `claude` or `codex`	Required for LLM-assisted commands (`analyze`, `plan`, `build`, `document`, `rigging`).

Check Python:

```
python3 --version   # must report 3.11 or higher
```

Drydock is for **subscription-authenticated CLI users**. It never uses API-key-backed model calls and never bills per token. LLM-assisted commands shell out to a provider CLI that you have already installed and authenticated:

- `claude` — Anthropic Claude subscription CLI.
- `codex` — OpenAI Codex subscription CLI.

Deterministic commands (`status`, `validate`, `document assemble`, `publish`, `shipslog`) do not call an LLM and work without a provider CLI.

2. Install the command

Recommended — isolated tool install:

```
uv tool install drydock-sdd
```

Alternative with `pipx`:

```
pipx install drydock-sdd
```

Into an active virtual environment:

```
python -m pip install drydock-sdd
```

`uv tool` and `pipx` install Drydock in a dedicated environment and put a command wrapper on your `PATH`; they are the right choice for an interactive CLI. `pip` installs Drydock into whatever virtual environment is active.

Verify:

```
drydock --version
drydock --help
```

Optional: PDF publishing

`drydock publish --pdf` and `drydock document --pdf` need the `pdf` extra plus a local Chromium download:

```
uv tool install "drydock-sdd[pdf]"
playwright install chromium
```

Optional: Claude Code skills

Drydock ships two Claude Code skills for the Loop phase — `/refit` (design discussion captured to a notes file) and `/apply-refit` (turn captured decisions into change tickets). Install them into `~/.claude/skills/`:

```
python -c "import shutil, pathlib; from drydock.paths import get_rigging_root; \
dest = pathlib.Path.home() / '.claude' / 'skills'; \
[shutil.copytree(s, dest / s.name, dirs_exist_ok=True) \
 for s in (get_rigging_root() / 'skills').iterdir()]"
```

See Drydock_SKILLS.md for usage.

3. Select and authenticate a provider

Point Drydock at the provider CLI you use, then confirm that CLI is installed, authenticated, and on `PATH`:

```
drydock config set llm_provider claude    # or: codex
claude --version                          # must resolve and be logged in
```

If the provider CLI is missing or unauthenticated, deterministic commands still work but LLM-assisted commands will fail.

4. Configure the workspace

Drydock needs two directories: a **workspace** that holds Targets and logs, and a **build directory** where generated applications are written. Both must exist before you configure them.

```
export PROJECTS="$HOME/projects"
mkdir -p "$PROJECTS/drydock"

drydock config set drydock_workspace "$PROJECTS/drydock"
drydock config set drydock_build_directory "$PROJECTS"
drydock config show
```

Resulting layout:

```
$PROJECTS/
├── drydock/            # Drydock workspace
│   ├── targets/        # Created by `drydock init`
│   └── logs/           # Created when commands run
└── <Target>/           # Generated application output
```

Do not create `targets/` by hand — `drydock init` creates it.

Where configuration lives

`drydock config set` writes per-user configuration; it never modifies the installed package.

Platform	Configuration file
Linux and macOS	`$XDG_CONFIG_HOME/drydock/.env`, or `~/.config/drydock/.env` when `XDG_CONFIG_HOME` is unset
Windows	`%APPDATA%\drydock\.env`

Environment variables override saved configuration for a single command — useful for CI or switching workspaces temporarily:

```
DRYDOCK_WORKSPACE="$PROJECTS/client-a/drydock" drydock status
```

CLI key	Environment variable	Purpose
`drydock_workspace`	`DRYDOCK_WORKSPACE`	Workspace containing `targets/` and logs
`drydock_build_directory`	`DRYDOCK_BUILD_DIRECTORY`	Root where generated applications are written
`drydock_model`	`DRYDOCK_MODEL`	Default model for LLM-assisted commands
`llm_provider`	`LLM_PROVIDER`	Subscription CLI provider: `claude` or `codex`

CLI key	Environment variable	Purpose
`prompt_warn_tokens`	`PROMPT_WARN_TOKENS`	Prompt-size warning threshold in tokens
`quarterdeck_port`	`QUARTERDECK_PORT`	Default QuarterDeck port

5. Create your first Target

```
drydock init Example
drydock status
```

This creates the user-owned Target workspace:

```
$PROJECTS/drydock/targets/Example/
├── blueprint/
│   └── sources/
├── evidence/
├── logs/
├── QuarterDeck/
│   └── console.yaml
└── METADATA.md
```

The Target name is a logical identifier, not a path: no path separators, no `..`.

6. Run the SAIL loop

Import your source material, analyze it into stories and acceptance criteria, review in the QuarterDeck, plan a dependency graph, then build and verify one frontier at a time:

```
drydock import Example ./notes --format markdown
drydock analyze Example
drydock run quarterdeck Example       # opens the web review surface
drydock plan Example

drydock build Example
drydock build status Example
drydock build verify Example <step-id>
```

The Target lives under `$DRYDOCK_WORKSPACE/targets/Example/`. The generated application is written under `$DRYDOCK_BUILD_DIRECTORY/Example/`.

7. What installation provides

The `drydock-sdd` distribution installs the `drydock` package and entry point, Drydock's versioned Rigging rules and Blueprint templates, versioned prompt contracts, the QuarterDeck runtime, the read-only canonical

product specification at `drydock/resources/docs/Drydock_Specification.md`, and required Python dependencies.

Installation does **not** create or own a project. Targets, Blueprints, manifests, evidence, Sea Trials, Soundings, logs, QuarterDeck state, and generated application code are created under your configured directories and remain owned by you.

Packaged resources are coupled to the installed release: upgrading `drydock-sdd` upgrades the command and its rules, templates, prompts, and QuarterDeck runtime together. Existing Targets are never rewritten by installation or upgrade.

8. Upgrade and removal

```
uv tool upgrade drydock-sdd
# or: pipx upgrade drydock-sdd
# or, in an active virtual environment: python -m pip install --upgrade drydock-sdd
```

Uninstalling removes the command and packaged resources only. It does not remove your configuration file, `$PROJECTS/drydock`, or generated applications.

9. Install from source (contributors)

```
git clone https://github.com/webcloudstudio/Drydock.git
cd Drydock
uv venv
uv pip install -e ".[dev]"
drydock --help
```

Troubleshooting

Symptom	Fix
`drydock: command not found`	Ensure the `uv`/`pipx` bin directory is on `PATH` (`uv tool update-shell` or `pipx ensurepath`), then open a new shell.
LLM-assisted command fails immediately	Confirm the provider CLI resolves and is authenticated: `claude --version` (or `codex --version`), and that `llm_provider` matches.
`workspace not configured`	Run the `drydock config set drydock_workspace ...` step and confirm with `drydock config show`.
`--pdf` fails	Install the `pdf` extra and run `playwright install chromium`.
Wrong workspace used	An environment variable (`DRYDOCK_WORKSPACE`, etc.) is overriding config for that shell — unset it.