FAQ

- Is Docker Sandboxes free? Can I use it commercially?
- Why do I need to sign in?
- Can I enforce sandbox policies across my organization?
- Which domains do I need to allow for Docker Sandboxes to work?
- Does the CLI collect telemetry?
- How do I set custom environment variables inside a sandbox?
- Why do agents run without approval prompts?
- How do I know if my agent is running in a sandbox?
- Why doesn't the sandbox use my user-level agent configuration?
- Can I paste images into an agent?
- Can I use Docker Sandboxes on headless Linux?

Host integration and workspace instructions on this page describe local sandboxes. See Local and cloud differences before adapting those workflows to the cloud.

Is Docker Sandboxes free? Can I use it commercially?

The `sbx` CLI and local sandbox compute are free to use, including for commercial and professional work. Cloud sandbox compute uses a pay-as-you-go subscription. Model-provider charges are separate from sandbox compute.

Organization governance for local sandboxes includes centrally managed network, filesystem, and MCP policies, sign-in enforcement, and audit logs. These organization governance features require a separate paid subscription — contact Docker Sales to get started.

Why do I need to sign in?

Docker Sandboxes is built around the idea that you and your agents are a team. Signing in gives each sandbox a verified identity, which lets Docker:

- **Tie sandboxes to a real person.** Governance matters when agents can build containers, install packages, and push code. Your Docker identity is the anchor.
- **Enable team features.** Team-scale features like organization governance, shared environments, and audit logs need a concept of "who," and adding that later would be worse for everyone.
- **Authenticate against Docker infrastructure.** Sandboxes pull images, run daemons, and talk to Docker services. A Docker account authenticates those requests.

Your Docker account email is only used for authentication, not marketing.

Can I enforce sandbox policies across my organization?

Yes. Admins can centrally manage network, filesystem, and MCP policies. These controls apply to local sandboxes in the organization. When organization governance is active, only organization allow rules grant access: local allow rules set with `sbx policy` are no longer evaluated, while local deny rules still apply on top.

See Organization policies. This feature requires a separate paid subscription — contact Docker Sales to get started.

Cloud sandboxes use separate network policy configuration. See Cloud network policy for cloud controls.

Which domains do I need to allow for Docker Sandboxes to work?

If your organization restricts outbound network access with a firewall or proxy, add the following domains to your allowlist so that `sbx` can authenticate, pull images, and report diagnostics for local sandboxes. Cloud operations also connect to `https://api.sandboxes-cloud.docker.com`.

Domain	Description
https://login.docker.com	Authentication
https://hub.docker.com	Docker Hub
https://api.docker.com	Docker API
https://marlin-2.docker.com	Telemetry
https://marlin-api.docker.com	Telemetry
https://registry-1.docker.io	Docker pull/push
https://auth.docker.io	Registry authentication
https://dhi.io	Docker Hardened Images
https://sbx-diagnostics.s3.us-east-1.amazonaws.com	Diagnostics upload

Does the CLI collect telemetry?

The `sbx` CLI collects basic usage data about CLI invocations:

- Which command you ran
- Whether it succeeded or failed
- How long it took
- If you're signed in, your Docker username is included

CLI usage telemetry does not include your prompts or code. Cloud sandboxes execute on Docker-managed infrastructure, so files you transfer to them are stored in the cloud.

To opt out of CLI usage analytics, set the `SBX_NO_TELEMETRY` environment variable:

```
$ export SBX_NO_TELEMETRY=1
```

How do I set custom environment variables inside a sandbox?

Starting with `sbx` version 0.39.0, use `-e`/`--env` or `--env-file` with `sbx run` and `sbx create`. See Set environment variables for syntax, precedence rules, persistent configuration for an existing sandbox, and guidance for credentials.

Variables in `/etc/sandbox-persistent.sh` are available to interactive sessions and agents started with `sbx run`. A variable only takes effect for sessions and agents started after it's added. Restart a running agent, or stop and start the sandbox, to pick up the new value.

Why do agents run without approval prompts?

The sandbox itself is the safety boundary. Because agents run inside an isolated microVM with network policies, credential isolation, and no access to your host system outside explicitly shared paths, the usual reasons for approval prompts (preventing destructive commands, network access, file modifications) are handled by the sandbox isolation layers instead.

If you prefer to re-enable approval prompts, change the permission mode inside the session. Most agents let you switch permission modes after startup. In Claude Code, use the `/permissions` command to change the mode interactively.

To make approval prompts the default for every session, create a v2 sandbox kit that extends the built-in agent and changes its launch options. See Fork an existing agent for a complete example.

For an environment built entirely with v3 kits, set the launch command in the workload's Dockerfile. See Build a v3 agent kit.

How do I know if my agent is running in a sandbox?

Ask the agent. The agent can see whether or not it's running inside a sandbox. In Claude Code, use the `/btw` slash command to ask without interrupting an in-progress task:

```
/btw are you running in a sandbox?
```

Why doesn't the sandbox use my user-level agent configuration?

Local sandboxes don't import your complete user-level agent configuration. Hooks, settings, and other files under directories such as `~/.claude` remain on the host. Project-level configuration in the working directory remains available inside the sandbox.

Shared agent skills are the exception. Use `sbx skills add` to install skills from a Git repository, or run `sbx skills import` to copy skills from supported host directories. `sbx` keeps the skills in a persistent store shared with sandboxes. See Share agent skills for repository management, supported host directories, mount behavior, and per-sandbox opt-out.

Keep project-specific skills and other agent configuration in the project itself. This versions the configuration alongside the code. Don't use symlinks to host paths because a sandboxed agent can't follow them outside the sandbox.

Can I paste images into an agent?

In local sandboxes, image paste is off by default. Text paste works because the terminal sends it directly. Pasting an image or screenshot with `Ctrl+V` is different: the agent reads it from your host clipboard, and the sandbox blocks that access unless you opt in.

Turn on `clipboard.imagePaste`:

```
$ sbx settings set clipboard.imagePaste true
```

`Ctrl+V` then pastes host images into agents that read the clipboard, including Claude Code and Codex. The setting takes effect within a few seconds, even for running sandboxes.

This is opt-in because it relaxes the sandbox's isolation: when enabled, a process inside the sandbox can read your host clipboard through the host-side proxy. The exposure is narrow — reads happen only on a paste, return image data only (`image/png`), and clipboard content is never cached or logged — but it's still host data crossing into the sandbox, so it stays off until you turn it on.

To turn it back off:

```
$ sbx settings set clipboard.imagePaste false
```

Can I use Docker Sandboxes on headless Linux?

Yes. For local sandboxes on Linux, `sbx` stores secrets in the Secret Service exposed by your desktop keyring, such as GNOME Keyring or KDE Wallet. Headless servers and some WSL setups have no running Secret Service, so `sbx` falls back to a file under `$XDG_CONFIG_HOME/com.docker.sandboxes`, which defaults to `~/.config/com.docker.sandboxes` when `$XDG_CONFIG_HOME` is unset. No setup is required. When you store a secret on such a host, `sbx` prints a notice:

```
No keychain detected - this secret will be stored on disk, protected by file permissions
rather than a password
```

`sbx` stores the file in a directory with `0700` permissions, the same file-permission model used for `~/.docker/config.json`. Any user or process that can read the file can retrieve the stored credentials, so treat the directory as sensitive. Where available, prefer a keychain, which mediates access per application.

To keep secrets in a keyring instead, run a Secret Service on the host before storing them: install `gnome-keyring` and start `dbus-run-session`, or run the keyring daemon under a login session that unlocks it. Once a working Secret Service is available, `sbx` stores new secrets in the keychain again. For where each platform keeps secrets, see Where secrets are stored.

□□□ #2
□ □□□ □□□ 2026-10-31 19:00:34 UTC
□ □□□ □□□ 2026-10-31 19:00:35 UTC