Security and sandboxing

Sandboxing

OpenClaw can run **tools inside Docker containers** to reduce blast radius. This is **optional** and controlled by configuration (`agents.defaults.sandbox` or `agents.list[].sandbox`). If sandboxing is off, tools run on the host. The Gateway stays on the host; tool execution runs in an isolated sandbox when enabled.

This is not a perfect security boundary, but it materially limits filesystem and process access when the model does something dumb.

What gets sandboxed

- Tool execution (`exec` , `read` , `write` , `edit` , `apply_patch` , `process` , etc.).
- Optional sandboxed browser (`agents.defaults.sandbox.browser`).
  - By default, the sandbox browser auto-starts (ensures CDP is reachable) when the browser tool needs it. Configure via `agents.defaults.sandbox.browser.autoStart` and `agents.defaults.sandbox.browser.autoStartTimeoutMs` .
  - `agents.defaults.sandbox.browser.allowHostControl` lets sandboxed sessions target the host browser explicitly.
  - Optional allowlists gate `target: "custom"` : `allowedControlUrls` , `allowedControlHosts` , `allowedControlPorts` .

Not sandboxed:

The Gateway process itself.

Any tool explicitly allowed to run on the host (e.g. `tools.elevated`).

Elevated exec runs on the host and bypasses sandboxing.

If sandboxing is off, `tools.elevated` does not change execution (already on host). See **Elevated Mode**.

Modes

`agents.defaults.sandbox.mode` controls **when** sandboxing is used:

`"off"`: no sandboxing.

`"non-main"`: sandbox only **non-main** sessions (default if you want normal chats on host).

`"all"`: every session runs in a sandbox. Note: `"non-main"` is based on `session.mainKey` (default `"main"`), not agent id. Group/channel sessions use their own keys, so they count as non-main and will be sandboxed.

Scope

`agents.defaults.sandbox.scope` controls **how many containers** are created:

`"session"` (default): one container per session.

`"agent"`: one container per agent.

`"shared"`: one container shared by all sandboxed sessions.

Workspace access

`agents.defaults.sandbox.workspaceAccess` controls **what the sandbox can see**:

`"none"` (default): tools see a sandbox workspace under `~/.openclaw/sandboxes`.

`"ro"`: mounts the agent workspace read-only at `/agent` (disables `write` / `edit` / `apply_patch`).

`"rw"`: mounts the agent workspace read/write at `/workspace`.

Inbound media is copied into the active sandbox workspace (`media/inbound/*`). Skills note: the `read` tool is sandbox-rooted. With `workspaceAccess: "none"`, OpenClaw mirrors eligible skills into the sandbox workspace (`.../skills`) so they can be read. With `"rw"`, workspace skills are readable from `/workspace/skills`.

Custom bind mounts

`agents.defaults.sandbox.docker.binds` mounts additional host directories into the container. Format: `host:container:mode` (e.g., `"/home/user/source:/source:rw"`).

Global and per-agent binds are **merged** (not replaced). Under `scope: "shared"`, per-agent binds are ignored.

`agents.defaults.sandbox.browser.binds` mounts additional host directories into the **sandbox browser** container only.

- When set (including `[]`), it replaces `agents.defaults.sandbox.docker.binds` for the browser container.
- When omitted, the browser container falls back to `agents.defaults.sandbox.docker.binds` (backwards compatible).

Example (read-only source + an extra data directory):

```
{
  agents: {
    defaults: {
      sandbox: {
        docker: {
          binds: ["/home/user/source:/source:ro", "/var/data/myapp:/data:ro"],
        },
      },
    },
    list: [
      {
        id: "build",
        sandbox: {
          docker: {
            binds: ["/mnt/cache:/cache:rw"],
          },
        },
      },
    ],
  },
}
```

Security notes:

- Binds bypass the sandbox filesystem: they expose host paths with whatever mode you set (`:ro` or `:rw`).
- OpenClaw blocks dangerous bind sources (for example: `docker.sock` , `/etc` , `/proc` , `/sys` , `/dev` , and parent mounts that would expose them).
- Sensitive mounts (secrets, SSH keys, service credentials) should be `:ro` unless absolutely required.
- Combine with `workspaceAccess: "ro"` if you only need read access to the workspace; bind modes stay independent.
- See for how binds interact with tool policy and elevated exec.

Images + setup

Default image: `openclaw-sandbox:bookworm-slim`

Build it once:

```
scripts/sandbox-setup.sh
```

Note: the default image does **not** include Node. If a skill needs Node (or other runtimes), either bake a custom image or install via `sandbox.docker.setupCommand` (requires network egress + writable root + root user).

Sandboxed browser image:

```
scripts/sandbox-browser-setup.sh
```

By default, sandbox containers run with **no network**. Override with `agents.defaults.sandbox.docker.network`.

Docker installs and the containerized gateway live here:

setupCommand (one-time container setup)

`setupCommand` runs **once** after the sandbox container is created (not on every run). It executes inside the container via `sh -lc`.

Paths:

- Global: `agents.defaults.sandbox.docker.setupCommand`
- Per-agent: `agents.list[].sandbox.docker.setupCommand`

Common pitfalls:

Default `docker.network` is `"none"` (no egress), so package installs will fail.

`readOnlyRoot: true` prevents writes; set `readOnlyRoot: false` or bake a custom image.

`user` must be root for package installs (omit `user` or set `user: "0:0"`).

Sandbox exec does **not** inherit host `process.env`. Use `agents.defaults.sandbox.docker.env` (or a custom image) for skill API keys.

Tool policy + escape hatches

Tool allow/deny policies still apply before sandbox rules. If a tool is denied globally or per-agent, sandboxing doesn't bring it back.

`tools.elevated` is an explicit escape hatch that runs `exec` on the host. `/exec` directives only apply for authorized senders and persist per session; to hard-disable `exec`, use tool policy deny (see **Sandbox vs Tool Policy vs Elevated**).

Debugging:

Use `openclaw sandbox explain` to inspect effective sandbox mode, tool policy, and fix-it config keys.

See **Sandbox vs Tool Policy vs Elevated** for the "why is this blocked?" mental model. Keep it locked down.

Multi-agent overrides

Each agent can override sandbox + tools: `agents.list[].sandbox` and `agents.list[].tools` (plus `agents.list[].tools.sandbox.tools` for sandbox tool policy). See **Multi-Agent Sandbox & Tools** for precedence.

Minimal enable example

```
{
  agents: {
    defaults: {
      sandbox: {
        mode: "non-main",
        scope: "session",
        workspaceAccess: "none",
      },
    },
  },
}
```

Related docs

‹ Security

Sandbox vs Tool Policy vs Elevated ›

Powered by mintlify