Technical reference

Onboarding Wizard Reference

This is the full reference for the `openclaw onboard` CLI wizard. For a high-level overview, see Onboarding Wizard.

Flow details (local mode)

1. **Existing config detection**

   If `~/.openclaw/openclaw.json` exists, choose **Keep / Modify / Reset**.

   Re-running the wizard does **not** wipe anything unless you explicitly choose **Reset** (or pass `--reset`).

   If the config is invalid or contains legacy keys, the wizard stops and asks you to run `openclaw doctor` before continuing.

   Reset uses `trash` (never `rm`) and offers scopes:

   - Config only
   - Config + credentials + sessions
   - Full reset (also removes workspace)

2. **Model/Auth**

   Anthropic API key (recommended): uses `ANTHROPIC_API_KEY` if present or prompts for a key, then saves it for daemon use.

Anthropic OAuth (Claude Code CLI): on macOS the wizard checks Keychain item “Claude Code-credentials” (choose “Always Allow” so launchd starts don’t block); on Linux/Windows it reuses `~/.claude/.credentials.json` if present.

Anthropic token (paste setup-token): run `claude setup-token` on any machine, then paste the token (you can name it; blank = default).

OpenAI Code (Codex) subscription (Codex CLI): if `~/.codex/auth.json` exists, the wizard can reuse it.

OpenAI Code (Codex) subscription (OAuth): browser flow; paste the `code#state`.

> Sets `agents.defaults.model` to `openai-codex/gpt-5.2` when model is unset or `openai/*`.

OpenAI API key: uses `OPENAI_API_KEY` if present or prompts for a key, then saves it to `~/.openclaw/.env` so launchd can read it.

xAI (Grok) API key: prompts for `XAI_API_KEY` and configures xAI as a model provider.

OpenCode Zen (multi-model proxy): prompts for `OPENCODE_API_KEY` (or `OPENCODE_ZEN_API_KEY`, get it at https://opencode.ai/auth).

API key: stores the key for you.

Vercel AI Gateway (multi-model proxy): prompts for `AI_GATEWAY_API_KEY`.

More detail: Vercel AI Gateway

Cloudflare AI Gateway: prompts for Account ID, Gateway ID, and `CLOUDFLARE_AI_GATEWAY_API_KEY`.

More detail: Cloudflare AI Gateway

MiniMax M2.1: config is auto-written.

More detail: MiniMax

Synthetic (Anthropic-compatible): prompts for `SYNTHETIC_API_KEY`.

More detail: **Synthetic**

Moonshot (Kimi K2): config is auto-written.

Kimi Coding: config is auto-written.

More detail: **Moonshot AI (Kimi + Kimi Coding)**

Skip: no auth configured yet.

Pick a default model from detected options (or enter provider/model manually).

Wizard runs a model check and warns if the configured model is unknown or missing auth.

OAuth credentials live in `~/.openclaw/credentials/oauth.json`; auth profiles live in `~/.openclaw/agents/<agentId>/agent/auth-profiles.json` (API keys + OAuth).

More detail: **/concepts/oauth**

> Headless/server tip: complete OAuth on a machine with a browser, then copy `~/.openclaw/credentials/oauth.json` (or `$OPENCLAW_STATE_DIR/credentials/oauth.json`) to the gateway host.

3 Workspace

Default `~/.openclaw/workspace` (configurable).

Seeds the workspace files needed for the agent bootstrap ritual.

Full workspace layout + backup guide: **Agent workspace**

4 Gateway

Port, bind, auth mode, tailscale exposure.

Auth recommendation: keep **Token** even for loopback so local WS clients must authenticate.

Disable auth only if you fully trust every local process.

Non-loopback binds still require auth.

5 Channels

WhatsApp: optional QR login.

Telegram: bot token.

Discord: bot token.

Google Chat: service account JSON + webhook audience.

Mattermost (plugin): bot token + base URL.

Signal: optional `signal-cli` install + account config.

BlueBubbles: **recommended for iMessage**; server URL + password + webhook.

iMessage: legacy `imsg` CLI path + DB access.

DM security: default is pairing. First DM sends a code; approve via `openclaw pairing approve <channel> <code>` or use allowlists.

6 Daemon install

macOS: LaunchAgent

- Requires a logged-in user session; for headless, use a custom LaunchDaemon (not shipped).

Linux (and Windows via WSL2): systemd user unit

- Wizard attempts to enable lingering via `loginctl enable-linger <user>` so the Gateway stays up after logout.
- May prompt for sudo (writes `/var/lib/systemd/linger`); it tries without sudo first.

Runtime selection: Node (recommended; required for WhatsApp/Telegram). Bun is **not recommended**.

Health check

Starts the Gateway (if needed) and runs `openclaw health`.

Tip: `openclaw status --deep` adds gateway health probes to status output (requires a reachable gateway).

8 Skills (recommended)

Reads the available skills and checks requirements.

Lets you choose a node manager: **npm / pnpm** (bun not recommended).

Installs optional dependencies (some use Homebrew on macOS).

9 Finish

Summary + next steps, including iOS/Android/macOS apps for extra features.

> If no GUI is detected, the wizard prints SSH port-forward instructions for the Control UI instead of opening a browser. If the Control UI assets are missing, the wizard attempts to build them; fallback is `pnpm ui:build` (auto-installs UI deps).

Non-interactive mode

Use `--non-interactive` to automate or script onboarding:

```
openclaw onboard --non-interactive \
  --mode local \
  --auth-choice apiKey \
  --anthropic-api-key "$ANTHROPIC_API_KEY" \
  --gateway-port 18789 \
  --gateway-bind loopback \
  --install-daemon \
  --daemon-runtime node \
  --skip-skills
```

Add `--json` for a machine-readable summary.

> `--json` does **not** imply non-interactive mode. Use `--non-interactive` (and `--workspace`) for scripts.

Gemini example
Z.AI example
Vercel AI Gateway example
Cloudflare AI Gateway example
Moonshot example
Synthetic example
OpenCode Zen example

Add agent (non-interactive)

```
openclaw agents add work \
  --workspace ~/.openclaw/workspace-work \
  --model openai/gpt-5.2 \
  --bind whatsapp:biz \
  --non-interactive \
  --json
```

Gateway wizard RPC

The Gateway exposes the wizard flow over RPC (`wizard.start` , `wizard.next` , `wizard.cancel` , `wizard.status`). Clients (macOS app, Control UI) can render steps without re-implementing onboarding logic.

Signal setup (signal-cli)

The wizard can install `signal-cli` from GitHub releases:

- Downloads the appropriate release asset.
- Stores it under `~/.openclaw/tools/signal-cli/<version>/` .
- Writes `channels.signal.cliPath` to your config.

Notes:

- JVM builds require **Java 21**.
- Native builds are used when available.
- Windows uses WSL2; signal-cli install follows the Linux flow inside WSL.

What the wizard writes

Typical fields in `~/.openclaw/openclaw.json` :

- `agents.defaults.workspace`

`agents.defaults.model` / `models.providers` (if Minimax chosen)

`gateway.*` (mode, bind, auth, tailscale)

`channels.telegram.botToken`, `channels.discord.token`, `channels.signal.*`, `channels.imessage.*`

Channel allowlists (Slack/Discord/Matrix/Microsoft Teams) when you opt in during the prompts (names resolve to IDs when possible).

`skills.install.nodeManager`

`wizard.lastRunAt`

`wizard.lastRunVersion`

`wizard.lastRunCommit`

`wizard.lastRunCommand`

`wizard.lastRunMode`

`openclaw agents add` writes `agents.list[]` and optional `bindings`.

WhatsApp credentials go under `~/.openclaw/credentials/whatsapp/<accountId>/`. Sessions are stored under `~/.openclaw/agents/<agentId>/sessions/`.

Some channels are delivered as plugins. When you pick one during onboarding, the wizard will prompt to install it (npm or a local path) before it can be configured.

Related docs

Wizard overview: **Onboarding Wizard**

macOS app onboarding: **Onboarding**

Config reference: **Gateway configuration**

Providers: **WhatsApp**, **Telegram**, **Discord**, **Google Chat**, **Signal**, **BlueBubbles** (iMessage), **iMessage** (legacy)

Skills: **Skills**, **Skills config**

‹ USER Token Use and Costs ›