Environment and debugging

Testing

OpenClaw has three Vitest suites (unit/integration, e2e, live) and a small set of Docker runners.

This doc is a “how we test” guide:

- What each suite covers (and what it deliberately does *not* cover)
- Which commands to run for common workflows (local, pre-push, debugging)
- How live tests discover credentials and select models/providers
- How to add regressions for real-world model/provider issues

Quick start

Most days:

- Full gate (expected before push): `pnpm build && pnpm check && pnpm test`

When you touch tests or want extra confidence:

- Coverage gate: `pnpm test:coverage`
- E2E suite: `pnpm test:e2e`

When debugging real providers/models (requires real creds):

- Live suite (models + gateway tool/image probes): `pnpm test:live`

Tip: when you only need one failing case, prefer narrowing live tests via the allowlist env vars described below.

Test suites (what runs where)

Think of the suites as “increasing realism” (and increasing flakiness/cost):

Unit / integration (default)

- Command: `pnpm test`
- Config: `scripts/test-parallel.mjs` (runs `vitest.unit.config.ts`, `vitest.extensions.config.ts`, `vitest.gateway.config.ts`)
- Files: `src/**/*.test.ts`, `extensions/**/*.test.ts`
- Scope:
  - Pure unit tests
  - In-process integration tests (gateway auth, routing, tooling, parsing, config)
  - Deterministic regressions for known bugs
- Expectations:
  - Runs in CI
  - No real keys required
  - Should be fast and stable
- Pool note:
  - OpenClaw uses Vitest `vmForks` on Node 22/23 for faster unit shards.
  - On Node 24+, OpenClaw automatically falls back to regular `forks` to avoid Node VM linking errors (`ERR_VM_MODULE_LINK_FAILURE` / `module is already linked`).

Override manually with `OPENCLAW_TEST_VM_FORKS=0` (force `forks`) or `OPENCLAW_TEST_VM_FORKS=1` (force `vmForks`).

›

E2E (gateway smoke)

Command: `pnpm test:e2e`

Config: `vitest.e2e.config.ts`

Files: `src/**/*.e2e.test.ts`

Runtime defaults:

- Uses Vitest `vmForks` for faster file startup.
- Uses adaptive workers (CI: 2-4, local: 4-8).
- Runs in silent mode by default to reduce console I/O overhead.

Useful overrides:

- `OPENCLAW_E2E_WORKERS=<n>` to force worker count (capped at 16).
- `OPENCLAW_E2E_VERBOSE=1` to re-enable verbose console output.

Scope:

- Multi-instance gateway end-to-end behavior
- WebSocket/HTTP surfaces, node pairing, and heavier networking

Expectations:

- Runs in CI (when enabled in the pipeline)
- No real keys required
- More moving parts than unit tests (can be slower)

Live (real providers + real models)

Command: `pnpm test:live`

Config: `vitest.live.config.ts`

Files: `src/**/*.live.test.ts`

Default: **enabled** by `pnpm test:live` (sets `OPENCLAW_LIVE_TEST=1`)

Scope:

- "Does this provider/model actually work *today* with real creds?"
- Catch provider format changes, tool-calling quirks, auth issues, and rate limit behavior

Expectations:

- Not CI-stable by design (real networks, real provider policies, quotas, outages)
- Costs money / uses rate limits
- Prefer running narrowed subsets instead of "everything"
- Live runs will source `~/.profile` to pick up missing API keys

API key rotation (provider-specific): set `*_API_KEYS` with comma/semicolon format or `*_API_KEY_1`, `*_API_KEY_2` (for example `OPENAI_API_KEYS`, `ANTHROPIC_API_KEYS`, `GEMINI_API_KEYS`) or per-live override via `OPENCLAW_LIVE_*_KEY`; tests retry on rate limit responses.

Which suite should I run?

Use this decision table:

- Editing logic/tests: run `pnpm test` (and `pnpm test:coverage` if you changed a lot)
- Touching gateway networking / WS protocol / pairing: add `pnpm test:e2e`
- Debugging "my bot is down" / provider-specific failures / tool calling: run a narrowed `pnpm test:live`

Live: model smoke (profile keys)

Live tests are split into two layers so we can isolate failures:

"Direct model" tells us the provider/model can answer at all with the given key. ›

"Gateway smoke" tells us the full gateway+agent pipeline works for that model (sessions, history, tools, sandbox policy, etc.).

Layer 1: Direct model completion (no gateway)

Test: `src/agents/models.profiles.live.test.ts`

Goal:

- Enumerate discovered models
- Use `getApiKeyForModel` to select models you have creds for
- Run a small completion per model (and targeted regressions where needed)

How to enable:

- `pnpm test:live` (or `OPENCLAW_LIVE_TEST=1` if invoking Vitest directly)

Set `OPENCLAW_LIVE_MODELS=modern` (or `all`, alias for modern) to actually run this suite; otherwise it skips to keep `pnpm test:live` focused on gateway smoke

How to select models:

- `OPENCLAW_LIVE_MODELS=modern` to run the modern allowlist (Opus/Sonnet/Haiku 4.5, GPT-5.x + Codex, Gemini 3, GLM 4.7, MiniMax M2.1, Grok 4)
- `OPENCLAW_LIVE_MODELS=all` is an alias for the modern allowlist
- or `OPENCLAW_LIVE_MODELS="openai/gpt-5.2,anthropic/claude-opus-4-6,..."` (comma allowlist)

How to select providers:

`OPENCLAW_LIVE_PROVIDERS="google,google-antigravity,google-gemini-cli"` (comma allowlist)

Where keys come from:

- By default: profile store and env fallbacks
- Set `OPENCLAW_LIVE_REQUIRE_PROFILE_KEYS=1` to enforce **profile store** only

Why this exists:

- Separates “provider API is broken / key is invalid” from “gateway agent pipeline is broken”
- Contains small, isolated regressions (example: OpenAI Responses/Codex Responses reasoning replay + tool-call flows)

Layer 2: Gateway + dev agent smoke (what “@openclaw” actually does)

Test: `src/gateway/gateway-models.profiles.live.test.ts`

Goal:

- Spin up an in-process gateway
- Create/patch a `agent:dev:*` session (model override per run)
- Iterate models-with-keys and assert:
  - “meaningful” response (no tools)
  - a real tool invocation works (read probe)
  - optional extra tool probes (exec+read probe)
  - OpenAI regression paths (tool-call-only → follow-up) keep working

Probe details (so you can explain failures quickly):

- `read` probe: the test writes a nonce file in the workspace and asks the agent to `read` it and echo the nonce back.
- `exec+read` probe: the test asks the agent to `exec` -write a nonce into a temp file, then `read` it back.

image probe: the test attaches a generated PNG (cat + randomized code) and expects the model to return `cat <CODE>` .

Implementation reference: `src/gateway/gateway-models.profiles.live.test.ts` and `src/gateway/live-image-probe.ts` .

How to enable:

`pnpm test:live` (or `OPENCLAW_LIVE_TEST=1` if invoking Vitest directly)

How to select models:

Default: modern allowlist (Opus/Sonnet/Haiku 4.5, GPT-5.x + Codex, Gemini 3, GLM 4.7, MiniMax M2.1, Grok 4)

`OPENCLAW_LIVE_GATEWAY_MODELS=all` is an alias for the modern allowlist

Or set `OPENCLAW_LIVE_GATEWAY_MODELS="provider/model"` (or comma list) to narrow

How to select providers (avoid “OpenRouter everything”):

`OPENCLAW_LIVE_GATEWAY_PROVIDERS="google,google-antigravity,google-gemini-cli,openai,anthropic,zai,minimax"` (comma allowlist)

Tool + image probes are always on in this live test:

`read` probe + `exec+read` probe (tool stress)

image probe runs when the model advertises image input support

Flow (high level):

Test generates a tiny PNG with “CAT” + random code (`src/gateway/live-image-probe.ts`)

Sends it via `agent` `attachments: [{ mimeType: "image/png", content: "<base64>" }]`

Gateway parses attachments into `images[]` (`src/gateway/server-methods/agent.ts` + `src/gateway/chat-attachments.ts`)

Embedded agent forwards a multimodal user message to the model

Assertion: reply contains `cat` + the code (OCR tolerance: minor mistakes allowed)

Tip: to see what you can test on your machine (and the exact `provider/model` ids), run:

```
openclaw models list
openclaw models list --json
```

Live: Anthropic setup-token smoke

Test: `src/agents/anthropic.setup-token.live.test.ts`

Goal: verify Claude Code CLI setup-token (or a pasted setup-token profile) can complete an Anthropic prompt.

Enable:

`pnpm test:live` (or `OPENCLAW_LIVE_TEST=1` if invoking Vitest directly)

`OPENCLAW_LIVE_SETUP_TOKEN=1`

Token sources (pick one):

Profile: `OPENCLAW_LIVE_SETUP_TOKEN_PROFILE=anthropic:setup-token-test`

Raw token: `OPENCLAW_LIVE_SETUP_TOKEN_VALUE=sk-ant-oat01-...`

Model override (optional):

`OPENCLAW_LIVE_SETUP_TOKEN_MODEL=anthropic/claude-opus-4-6`

Setup example:

```
openclaw models auth paste-token --provider anthropic --profile-id ant
OPENCLAW_LIVE_SETUP_TOKEN=1 OPENCLAW_LIVE_SETUP_TOKEN_PROFILE=anthropic:setup-toke
```

Live: CLI backend smoke (Claude Code CLI or other local CLIs)

Test: `src/gateway/gateway-cli-backend.live.test.ts`

Goal: validate the Gateway + agent pipeline using a local CLI backend, without touching your default config.

Enable:

- `pnpm test:live` (or `OPENCLAW_LIVE_TEST=1` if invoking Vitest directly)
- `OPENCLAW_LIVE_CLI_BACKEND=1`

Defaults:

- Model: `claude-cli/claude-sonnet-4-6`
- Command: `claude`
- Args: `["-p","--output-format","json","--dangerously-skip-permissions"]`

Overrides (optional):

- `OPENCLAW_LIVE_CLI_BACKEND_MODEL="claude-cli/claude-opus-4-6"`
- `OPENCLAW_LIVE_CLI_BACKEND_MODEL="codex-cli/gpt-5.3-codex"`
- `OPENCLAW_LIVE_CLI_BACKEND_COMMAND="/full/path/to/claude"`
- `OPENCLAW_LIVE_CLI_BACKEND_ARGS='["-p","--output-format","json","--permission-mode","bypassPermissions"]'`
- `OPENCLAW_LIVE_CLI_BACKEND_CLEAR_ENV='["ANTHROPIC_API_KEY","ANTHROPIC_API_KEY_OLD"]'`
- `OPENCLAW_LIVE_CLI_BACKEND_IMAGE_PROBE=1` to send a real image attachment (paths are injected into the prompt).
- `OPENCLAW_LIVE_CLI_BACKEND_IMAGE_ARG="--image"` to pass image file paths as CLI args instead of prompt injection.
- `OPENCLAW_LIVE_CLI_BACKEND_IMAGE_MODE="repeat"` (or `"list"`) to control how image args are passed when `IMAGE_ARG` is set.
- `OPENCLAW_LIVE_CLI_BACKEND_RESUME_PROBE=1` to send a second turn and validate resume flow.

`OPENCLAW_LIVE_CLI_BACKEND_DISABLE_MCP_CONFIG=0` to keep Claude Code CLI MCP config enabled (default disables MCP config with a temporary empty file).

Example:

```
OPENCLAW_LIVE_CLI_BACKEND=1 \
  OPENCLAW_LIVE_CLI_BACKEND_MODEL="claude-cli/claude-sonnet-4-6" \
  pnpm test:live src/gateway/gateway-cli-backend.live.test.ts
```

Recommended live recipes

Narrow, explicit allowlists are fastest and least flaky:

- Single model, direct (no gateway):

  ```
  OPENCLAW_LIVE_MODELS="openai/gpt-5.2" pnpm test:live src/agents/models.profiles.live.test.ts
  ```

- Single model, gateway smoke:

  ```
  OPENCLAW_LIVE_GATEWAY_MODELS="openai/gpt-5.2" pnpm test:live src/gateway/gateway-models.profiles.live.test.ts
  ```

- Tool calling across several providers:

  ```
  OPENCLAW_LIVE_GATEWAY_MODELS="openai/gpt-5.2,anthropic/claude-opus-4-6,google/gemini-3-flash-preview,zai/glm-4.7,minimax/minimax-m2.1" pnpm test:live src/gateway/gateway-models.profiles.live.test.ts
  ```

- Google focus (Gemini API key + Antigravity):

  - Gemini (API key): `OPENCLAW_LIVE_GATEWAY_MODELS="google/gemini-3-flash-preview" pnpm test:live src/gateway/gateway-models.profiles.live.test.ts`
  - Antigravity (OAuth): `OPENCLAW_LIVE_GATEWAY_MODELS="google-antigravity/claude-opus-4-6-thinking,google-antigravity/gemini-3-pro-high" pnpm test:live src/gateway/gateway-models.profiles.live.test.ts`

Notes:

`google/...` uses the Gemini API (API key).

`google-antigravity/...` uses the Antigravity OAuth bridge (Cloud Code Assist-style agent endpoint).

`google-gemini-cli/...` uses the local Gemini CLI on your machine (separate auth + tooling quirks).

Gemini API vs Gemini CLI:

API: OpenClaw calls Google's hosted Gemini API over HTTP (API key / profile auth); this is what most users mean by "Gemini".

CLI: OpenClaw shells out to a local `gemini` binary; it has its own auth and can behave differently (streaming/tool support/version skew).

Live: model matrix (what we cover)

There is no fixed "CI model list" (live is opt-in), but these are the **recommended** models to cover regularly on a dev machine with keys.

Modern smoke set (tool calling + image)

This is the "common models" run we expect to keep working:

OpenAI (non-Codex): `openai/gpt-5.2` (optional: `openai/gpt-5.1`)

OpenAI Codex: `openai-codex/gpt-5.3-codex` (optional: `openai-codex/gpt-5.3-codex-codex`)

Anthropic: `anthropic/claude-opus-4-6` (or `anthropic/claude-sonnet-4-5`)

Google (Gemini API): `google/gemini-3-pro-preview` and `google/gemini-3-flash-preview` (avoid older Gemini 2.x models)

Google (Antigravity): `google-antigravity/claude-opus-4-6-thinking` and `google-antigravity/gemini-3-flash`

Z.AI (GLM): `zai/glm-4.7`

MiniMax: `minimax/minimax-m2.1`

Run gateway smoke with tools + image:

```
OPENCLAW_LIVE_GATEWAY_MODELS="openai/gpt-5.2,openai-codex/gpt-5.3-codex,anthropic/claude-opus-4-6,google/gemini-3-pro-preview,google/gemini-3-flash-preview,google-antigravity/claude-opus-4-6-thinking,google-antigravity/gemini-3-flash,zai/glm-4.7,minimax/minimax-m2.1" pnpm test:live src/gateway/gateway-models.profiles.live.test.ts
```

Baseline: tool calling (Read + optional Exec)

Pick at least one per provider family:

- OpenAI: `openai/gpt-5.2` (or `openai/gpt-5-mini`)
- Anthropic: `anthropic/claude-opus-4-6` (or `anthropic/claude-sonnet-4-5`)
- Google: `google/gemini-3-flash-preview` (or `google/gemini-3-pro-preview`)
- Z.AI (GLM): `zai/glm-4.7`
- MiniMax: `minimax/minimax-m2.1`

Optional additional coverage (nice to have):

- xAI: `xai/grok-4` (or latest available)
- Mistral: `mistral/`… (pick one “tools” capable model you have enabled)
- Cerebras: `cerebras/`… (if you have access)
- LM Studio: `lmstudio/`… (local; tool calling depends on API mode)

Vision: image send (attachment → multimodal message)

Include at least one image-capable model in `OPENCLAW_LIVE_GATEWAY_MODELS` (Claude/Gemini/OpenAI vision-capable variants, etc.) to exercise the image probe.

Aggregators / alternate gateways

If you have keys enabled, we also support testing via:

- OpenRouter: `openrouter/...` (hundreds of models; use `openclaw models scan` to find tool+image capable candidates)
- OpenCode Zen: `opencode/...` (auth via `OPENCODE_API_KEY` / `OPENCODE_ZEN_API_KEY`)

More providers you can include in the live matrix (if you have creds/config):

- Built-in: `openai`, `openai-codex`, `anthropic`, `google`, `google-vertex`, `google-antigravity`, `google-gemini-cli`, `zai`, `openrouter`, `opencode`, `xai`, `groq`, `cerebras`, `mistral`, `github-copilot`
- Via `models.providers` (custom endpoints): `minimax` (cloud/API), plus any OpenAI/Anthropic-compatible proxy (LM Studio, vLLM, LiteLLM, etc.)

Tip: don't try to hardcode "all models" in docs. The authoritative list is whatever `discoverModels(...)` returns on your machine + whatever keys are available.

Credentials (never commit)

Live tests discover credentials the same way the CLI does. Practical implications:

- If the CLI works, live tests should find the same keys.
- If a live test says "no creds", debug the same way you'd debug `openclaw models list` / model selection.
- Profile store: `~/.openclaw/credentials/` (preferred; what "profile keys" means in the tests)
- Config: `~/.openclaw/openclaw.json` (or `OPENCLAW_CONFIG_PATH`)

If you want to rely on env keys (e.g. exported in your `~/.profile`), run local tests after `source ~/.profile`, or use the Docker runners below (they can mount `~/.profile` into the container).

Deepgram live (audio transcription)

- Test: `src/media-understanding/providers/deepgram/audio.live.test.ts`
- Enable: `DEEPGRAM_API_KEY=... DEEPGRAM_LIVE_TEST=1 pnpm test:live src/media-understanding/providers/deepgram/audio.live.test.ts`

Docker runners (optional “works in Linux” checks)

These run `pnpm test:live` inside the repo Docker image, mounting your local config dir and workspace (and sourcing `~/.profile` if mounted):

- Direct models: `pnpm test:docker:live-models` (script: `scripts/test-live-models-docker.sh`)
- Gateway + dev agent: `pnpm test:docker:live-gateway` (script: `scripts/test-live-gateway-models-docker.sh`)
- Onboarding wizard (TTY, full scaffolding): `pnpm test:docker:onboard` (script: `scripts/e2e/onboard-docker.sh`)
- Gateway networking (two containers, WS auth + health): `pnpm test:docker:gateway-network` (script: `scripts/e2e/gateway-network-docker.sh`)
- Plugins (custom extension load + registry smoke): `pnpm test:docker:plugins` (script: `scripts/e2e/plugins-docker.sh`)

Useful env vars:

- `OPENCLAW_CONFIG_DIR=...` (default: `~/.openclaw`) mounted to `/home/node/.openclaw`
- `OPENCLAW_WORKSPACE_DIR=...` (default: `~/.openclaw/workspace`) mounted to `/home/node/.openclaw/workspace`

- `OPENCLAW_PROFILE_FILE=...` (default: `~/.profile`) mounted to `/home/node/.profile` and sourced before running tests
- `OPENCLAW_LIVE_GATEWAY_MODELS=...` / `OPENCLAW_LIVE_MODELS=...` to narrow the run
- `OPENCLAW_LIVE_REQUIRE_PROFILE_KEYS=1` to ensure creds come from the profile store (not env)

Docs sanity

Run docs checks after doc edits: `pnpm docs:list`.

Offline regression (CI-safe)

These are “real pipeline” regressions without real providers:

- Gateway tool calling (mock OpenAI, real gateway + agent loop): `src/gateway/gateway.tool-calling.mock-openai.test.ts`
- Gateway wizard (WS `wizard.start` / `wizard.next`, writes config + auth enforced): `src/gateway/gateway.wizard.e2e.test.ts`

Agent reliability evals (skills)

We already have a few CI-safe tests that behave like “agent reliability evals”:

- Mock tool-calling through the real gateway + agent loop (`src/gateway/gateway.tool-calling.mock-openai.test.ts`).
- End-to-end wizard flows that validate session wiring and config effects (`src/gateway/gateway.wizard.e2e.test.ts`).

What’s still missing for skills (see **Skills**):

- **Decisioning:** when skills are listed in the prompt, does the agent pick the right skill (or avoid irrelevant ones)?
- **Compliance:** does the agent read `SKILL.md` before use and follow required steps/args?
- **Workflow contracts:** multi-turn scenarios that assert tool order, session history carryover, and sandbox boundaries.

Future evals should stay deterministic first:

- A scenario runner using mock providers to assert tool calls + order, skill file reads, and session wiring.
- A small suite of skill-focused scenarios (use vs avoid, gating, prompt injection).
- Optional live evals (opt-in, env-gated) only after the CI-safe suite is in place.

Adding regressions (guidance)

When you fix a provider/model issue discovered in live:

- Add a CI-safe regression if possible (mock/stub provider, or capture the exact request-shape transformation)
- If it's inherently live-only (rate limits, auth policies), keep the live test narrow and opt-in via env vars
- Prefer targeting the smallest layer that catches the bug:
  - provider request conversion/replay bug → direct models test
  - gateway session/history/tool pipeline bug → gateway live smoke or CI-safe gateway mock test

‹ Debugging Scripts ›