Configuration

Model Providers

This page covers **LLM/model providers** (not chat channels like WhatsApp/Telegram). For model selection rules, see /concepts/models.

Quick rules

- Model refs use `provider/model` (example: `opencode/claude-opus-4-6`).
- If you set `agents.defaults.models`, it becomes the allowlist.
- CLI helpers: `openclaw onboard`, `openclaw models list`, `openclaw models set <provider/model>`.

API key rotation

- Supports generic provider rotation for selected providers.
- Configure multiple keys via:
  - `OPENCLAW_LIVE_<PROVIDER>_KEY` (single live override, highest priority)
  - `<PROVIDER>_API_KEYS` (comma or semicolon list)
  - `<PROVIDER>_API_KEY` (primary key)
  - `<PROVIDER>_API_KEY_*` (numbered list, e.g. `<PROVIDER>_API_KEY_1`)
- For Google providers, `GOOGLE_API_KEY` is also included as fallback.
- Key selection order preserves priority and deduplicates values.

Requests are retried with the next key only on rate-limit responses (for example `429`, `rate_limit`, `quota`, `resource exhausted`).

Non-rate-limit failures fail immediately; no key rotation is attempted.

When all candidate keys fail, the final error is returned from the last attempt.

Built-in providers (pi-ai catalog)

OpenClaw ships with the pi-ai catalog. These providers require **no** `models.providers` config; just set auth + pick a model.

OpenAI

Provider: `openai`

Auth: `OPENAI_API_KEY`

Optional rotation: `OPENAI_API_KEYS`, `OPENAI_API_KEY_1`, `OPENAI_API_KEY_2`, plus `OPENCLAW_LIVE_OPENAI_KEY` (single override)

Example model: `openai/gpt-5.1-codex`

CLI: `openclaw onboard --auth-choice openai-api-key`

```
{
  agents: { defaults: { model: { primary: "openai/gpt-5.1-codex" } } },
}
```

Anthropic

Provider: `anthropic`

Auth: `ANTHROPIC_API_KEY` or `claude setup-token`

Optional rotation: `ANTHROPIC_API_KEYS`, `ANTHROPIC_API_KEY_1`, `ANTHROPIC_API_KEY_2`, plus `OPENCLAW_LIVE_ANTHROPIC_KEY` (single override)

Example model: `anthropic/claude-opus-4-6`

CLI: `openclaw onboard --auth-choice token` (paste setup-token) or `openclaw models auth paste-token --provider anthropic`

```
{
  agents: { defaults: { model: { primary: "anthropic/claude-opus-4-6" } } },
}
```

OpenAI Code (Codex)

Provider: `openai-codex`

Auth: OAuth (ChatGPT)

Example model: `openai-codex/gpt-5.3-codex`

CLI: `openclaw onboard --auth-choice openai-codex` or `openclaw models auth login --provider openai-codex`

```
{
  agents: { defaults: { model: { primary: "openai-codex/gpt-5.3-codex" } } },
}
```

OpenCode Zen

Provider: `opencode`

Auth: `OPENCODE_API_KEY` (or `OPENCODE_ZEN_API_KEY`)

Example model: `opencode/claude-opus-4-6`

CLI: `openclaw onboard --auth-choice opencode-zen`

```
{
  agents: { defaults: { model: { primary: "opencode/claude-opus-4-6" } } },
}
```

Google Gemini (API key)

- Provider: `google`
- Auth: `GEMINI_API_KEY`
- Optional rotation: `GEMINI_API_KEYS` , `GEMINI_API_KEY_1` , `GEMINI_API_KEY_2` , `GOOGLE_API_KEY` fallback, and `OPENCLAW_LIVE_GEMINI_KEY` (single override)
- Example model: `google/gemini-3-pro-preview`
- CLI: `openclaw onboard --auth-choice gemini-api-key`

Google Vertex, Antigravity, and Gemini CLI

- Providers: `google-vertex` , `google-antigravity` , `google-gemini-cli`
- Auth: Vertex uses gcloud ADC; Antigravity/Gemini CLI use their respective auth flows
- Antigravity OAuth is shipped as a bundled plugin (`google-antigravity-auth` , disabled by default).
  - Enable: `openclaw plugins enable google-antigravity-auth`
  - Login: `openclaw models auth login --provider google-antigravity --set-default`
- Gemini CLI OAuth is shipped as a bundled plugin (`google-gemini-cli-auth` , disabled by default).
  - Enable: `openclaw plugins enable google-gemini-cli-auth`
  - Login: `openclaw models auth login --provider google-gemini-cli --set-default`
  - Note: you do **not** paste a client id or secret into `openclaw.json` . The CLI login flow stores tokens in auth profiles on the gateway host.

Z.AI (GLM)

- Provider: `zai`
- Auth: `ZAI_API_KEY`
- Example model: `zai/glm-4.7`
- CLI: `openclaw onboard --auth-choice zai-api-key`
  - Aliases: `z.ai/*` and `z-ai/*` normalize to `zai/*`

Vercel AI Gateway

- Provider: `vercel-ai-gateway`
- Auth: `AI_GATEWAY_API_KEY`
- Example model: `vercel-ai-gateway/anthropic/claude-opus-4.6`
- CLI: `openclaw onboard --auth-choice ai-gateway-api-key`

Other built-in providers

- OpenRouter: `openrouter` (`OPENROUTER_API_KEY`)
- Example model: `openrouter/anthropic/claude-sonnet-4-5`
- xAI: `xai` (`XAI_API_KEY`)
- Groq: `groq` (`GROQ_API_KEY`)
- Cerebras: `cerebras` (`CEREBRAS_API_KEY`)
  - GLM models on Cerebras use ids `zai-glm-4.7` and `zai-glm-4.6` .
  - OpenAI-compatible base URL: `https://api.cerebras.ai/v1` .
- Mistral: `mistral` (`MISTRAL_API_KEY`)
- GitHub Copilot: `github-copilot` (`COPILOT_GITHUB_TOKEN` / `GH_TOKEN` / `GITHUB_TOKEN`)
- Hugging Face Inference: `huggingface` (`HUGGINGFACE_HUB_TOKEN` or `HF_TOKEN`) — OpenAI-compatible router; example model: `huggingface/deepseek-ai/DeepSeek-R1` ; CLI: `openclaw onboard --auth-choice huggingface-api-key` . See Hugging Face (Inference).

Providers via `models.providers` (custom/base URL)

Use `models.providers` (or `models.json`) to add **custom** providers or OpenAI/Anthropic-compatible proxies.

Moonshot AI (Kimi)

Moonshot uses OpenAI-compatible endpoints, so configure it as a custom provider:

- Provider: `moonshot`
- Auth: `MOONSHOT_API_KEY`
- Example model: `moonshot/kimi-k2.5`

Kimi K2 model IDs:

- `moonshot/kimi-k2.5`
- `moonshot/kimi-k2-0905-preview`
- `moonshot/kimi-k2-turbo-preview`
- `moonshot/kimi-k2-thinking`
- `moonshot/kimi-k2-thinking-turbo`

```
{
  agents: {
    defaults: { model: { primary: "moonshot/kimi-k2.5" } },
  },
  models: {
    mode: "merge",
    providers: {
      moonshot: {
        baseUrl: "https://api.moonshot.ai/v1",
        apiKey: "${MOONSHOT_API_KEY}",
        api: "openai-completions",
        models: [{ id: "kimi-k2.5", name: "Kimi K2.5" }],
      },
    },
  },
}
```

Kimi Coding

Kimi Coding uses Moonshot AI’s Anthropic-compatible endpoint:

- Provider: `kimi-coding`
- Auth: `KIMI_API_KEY`
- Example model: `kimi-coding/k2p5`

```
{
  env: { KIMI_API_KEY: "sk-..." },
  agents: {
    defaults: { model: { primary: "kimi-coding/k2p5" } },
  },
}
```

Qwen OAuth (free tier)

Qwen provides OAuth access to Qwen Coder + Vision via a device-code flow. Enable the bundled plugin, then log in:

```
openclaw plugins enable qwen-portal-auth
openclaw models auth login --provider qwen-portal --set-default
```

Model refs:

- `qwen-portal/coder-model`
- `qwen-portal/vision-model`

See for setup details and notes.

Synthetic

Synthetic provides Anthropic-compatible models behind the `synthetic` provider:

- Provider: `synthetic`
- Auth: `SYNTHETIC_API_KEY`
- Example model: `synthetic/hf:MiniMaxAI/MiniMax-M2.1`
- CLI: `openclaw onboard --auth-choice synthetic-api-key`

```
{
  agents: {
    defaults: { model: { primary: "synthetic/hf:MiniMaxAI/MiniMax-M2.1" } },
  },
  models: {
    mode: "merge",
    providers: {
      synthetic: {
        baseUrl: "https://api.synthetic.new/anthropic",
        apiKey: "${SYNTHETIC_API_KEY}",
        api: "anthropic-messages",
        models: [{ id: "hf:MiniMaxAI/MiniMax-M2.1", name: "MiniMax M2.1" }],
      },
    },
  },
}
```

MiniMax

MiniMax is configured via `models.providers` because it uses custom endpoints:

- MiniMax (Anthropic-compatible): `--auth-choice minimax-api`
- Auth: `MINIMAX_API_KEY`

See for setup details, model options, and config snippets.

Ollama

Ollama is a local LLM runtime that provides an OpenAI-compatible API:

- Provider: `ollama`
- Auth: None required (local server)
- Example model: `ollama/llama3.3`

Installation: https://ollama.ai

```
# Install Ollama, then pull a model:
ollama pull llama3.3
```

```
{
  agents: {
    defaults: { model: { primary: "ollama/llama3.3" } },
  },
}
```

Ollama is automatically detected when running locally at `http://127.0.0.1:11434/v1` . See for model recommendations and custom configuration.

vLLM

vLLM is a local (or self-hosted) OpenAI-compatible server:

- Provider: `vllm`
- Auth: Optional (depends on your server)
- Default base URL: `http://127.0.0.1:8000/v1`

To opt in to auto-discovery locally (any value works if your server doesn't enforce auth):

```
export VLLM_API_KEY="vllm-local"
```

Then set a model (replace with one of the IDs returned by `/v1/models`):

```
{
  agents: {
    defaults: { model: { primary: "vllm/your-model-id" } },
  },
}
```

See for details.

Local proxies (LM Studio, vLLM, LiteLLM, etc.)

Example (OpenAI-compatible):

```
{
  agents: {
    defaults: {
      model: { primary: "lmstudio/minimax-m2.1-gs32" },
      models: { "lmstudio/minimax-m2.1-gs32": { alias: "Minimax" } },
    },
  },
  models: {
    providers: {
      lmstudio: {
        baseUrl: "http://localhost:1234/v1",
        apiKey: "LMSTUDIO_KEY",
        api: "openai-completions",
        models: [
          {
            id: "minimax-m2.1-gs32",
            name: "MiniMax M2.1",
            reasoning: false,
            input: ["text"],
            cost: { input: 0, output: 0, cacheRead: 0, cacheWrite: 0 },
            contextWindow: 200000,
            maxTokens: 8192,
          },
        ],
      },
    },
  },
}
```

Notes:

- For custom providers, `reasoning`, `input`, `cost`, `contextWindow`, and `maxTokens` are optional. When omitted, OpenClaw defaults to:
  - `reasoning: false`
  - `input: ["text"]`
  - `cost: { input: 0, output: 0, cacheRead: 0, cacheWrite: 0 }`

```
contextWindow: 200000
maxTokens: 8192
```

Recommended: set explicit values that match your proxy/model limits.

CLI examples

```
openclaw onboard --auth-choice opencode-zen
openclaw models set opencode/claude-opus-4-6
openclaw models list
```

See also: for full configuration examples.

‹ Models CLI Model Failover ›