OpenClaw Master Setup & Workflow Guide

> **Version:** 2026 Edition

> **Framework:** OpenClaw (open-source AI agent framework)

> **Audience:** First-time installers through multi-agent power users

---

Table of Contents

---

1. Quickstart (10-Minute Path)

This gets you from zero to a working OpenClaw agent in under ten minutes. Security hardening and multi-agent patterns come later.

Prerequisites: Node.js >= 22.12.0 installed, an API key for at least one LLM provider (Anthropic, OpenAI, Google, etc.).

```
# 1. Install OpenClaw globally
npm install -g openclaw

# 2. Initialize (creates ~/.openclaw/ and config scaffold)
openclaw init

# 3. Edit the config to set your model
#    File: ~/.openclaw/openclaw.json
#    Minimum viable config:
```

```
{
  "agent": {
    "model": "anthropic/claude-opus-4-6"
  }
}
```

```
# 4. Start the gateway daemon
openclaw gateway start

# 5. Verify it's running
openclaw status
```

You now have a running OpenClaw gateway. To connect a messaging channel (Telegram, WhatsApp, Slack, Discord, etc.), see [Section 3.4](#34-channel-setup-messaging-integrations). To lock down security before doing anything else on a remote server, jump to [Section 7](#7-security-hardening).

---

2. Installation & Prerequisites

2.1 System Requirements

- **Node.js:** >= 22.12.0 (required; older versions have known security vulnerabilities)

- **OS:** macOS, Linux, or Windows (WSL2). Platform-specific guides exist for each.

- **Hardware:** Any modern machine works for API-backed models. Local model inference (e.g., Llama) requires significant RAM and compute; expect very slow token generation on underpowered hardware.

- **Docker:** Required only if you enable sandboxing (recommended for group/channel sessions).

2.2 Install Node.js

Check your current version:

```
node --version
```

If below 22.12.0, upgrade:

macOS (Homebrew):

```
brew update && brew upgrade node
```

Alternatively, download the installer from nodejs.org.

Ubuntu/Debian Linux:

```
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Windows: Download the latest LTS installer from nodejs.org.

Verify:

```
node --version
# Expected: v22.12.0 or higher
```

2.3 Install OpenClaw

```
npm install -g openclaw
openclaw init
```

`openclaw init` creates:

- `~/.openclaw/` — root configuration directory

- `~/.openclaw/openclaw.json` — main configuration file

- `~/.openclaw/credentials/` — stored credentials (channel logins, tokens)

- `~/.openclaw/workspace/` — agent workspace root

Verify:

```
openclaw status
```

2.4 API Key Setup

OpenClaw supports multiple LLM providers. Set your chosen provider's API key as an environment variable or in the configuration file. The model string in `openclaw.json` determines which provider is used.

Common model strings:

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

- `anthropic/claude-sonnet-4-5`

- `openai/gpt-5.2`

- `google/gemini-3-flash`

Set the relevant API key in your shell profile (`~/.bashrc`, `~/.zshrc`, or equivalent):

```
export ANTHROPIC_API_KEY="sk-ant-..."
# or
export OPENAI_API_KEY="sk-..."
# or
export GOOGLE_API_KEY="..."
```

Reload your shell:

```
source ~/.zshrc  # or source ~/.bashrc
```

---

3. Onboarding / Daemon / Gateway Setup

3.1 What the Gateway Does

The Gateway is the core process. It:

- Runs 24/7 as a background daemon

- Manages all active sessions (persistent conversations with context)

- Handles cron jobs (scheduled agent wakeups)

- Routes messages between channels (Telegram, Discord, etc.) and sessions

- Provides a WebSocket API for control and companion apps

3.2 Starting the Gateway

```
openclaw gateway start
```

Verify:

```
openclaw gateway status
```

Other gateway commands:

```
openclaw gateway restart    # Restart the daemon
```

3.3 Gateway Configuration

All configuration lives in `~/.openclaw/openclaw.json`. This file uses a JSON structure (with relaxed parsing that allows comments and trailing commas in some implementations).

Minimal config (model only):

```
{
  "agent": {
    "model": "anthropic/claude-opus-4-6"
  }
}
```

Gateway network binding:

```
{
  "gateway": {
    "bind": "loopback",
    "port": 18789
  }
}
```

- `"loopback"` binds to `127.0.0.1` only (recommended). Only the local machine can connect.

- `"lan"` or `"0.0.0.0"` binds to all interfaces. **Do not use on internet-facing servers without additional hardening.** See [Section 7](#7-security-hardening).

Gateway authentication:

```
{
  "gateway": {
    "auth": {
      "mode": "password"
    }
  }
}
```

Set the password via environment variable:

```
export CLAWDBOT_GATEWAY_PASSWORD="your-secure-password-here"
```

Or use token authentication:

```
export CLAWDBOT_GATEWAY_TOKEN="your-secure-random-token-here"
```

Generate a secure token:

```
openssl rand -hex 32
```

3.4 Channel Setup (Messaging Integrations)

OpenClaw supports WhatsApp, Telegram, Slack, Discord, Signal, iMessage (via BlueBubbles or legacy), Microsoft Teams, Google Chat, and WebChat. All channels are configured in `~/.openclaw/openclaw.json` under the `channels` key.

Telegram

Set the bot token via environment variable (takes precedence) or config:

```
export TELEGRAM_BOT_TOKEN="123456:ABCDEF"
```

Or in `openclaw.json`:

```
{
  "channels": {
    "telegram": {
      "botToken": "123456:ABCDEF"
    }
  }
}
```

Optional group settings:

- `channels.telegram.groups` — when set, acts as a group allowlist. Include `"*"` to allow all groups.
- `channels.telegram.groups."*".requireMention` — require @mention to activate in groups.
- `channels.telegram.allowFrom` — allowlist of user IDs.
- `channels.telegram.webhookUrl` + `channels.telegram.webhookSecret` — webhook mode.

WhatsApp

```
pnpm openclaw channels login
```

This links the device and stores credentials in `~/.openclaw/credentials`.

- `channels.whatsapp.allowFrom` — allowlist who can talk to the agent.
- `channels.whatsapp.groups` — when set, acts as a group allowlist. Include `"*"` to allow all.

Slack

Set via environment variables (preferred) or config:

```
export SLACK_BOT_TOKEN="xoxb-..."
export SLACK_APP_TOKEN="xapp-..."
```

Or: `channels.slack.botToken` + `channels.slack.appToken`.

Discord

```
export DISCORD_BOT_TOKEN="your-discord-bot-token"
```

Or in config:

```
{
  "channels": {
    "discord": {
      "token": "your-discord-bot-token"
    }
  }
}
```

Optional keys: `commands.native`, `commands.text`, `commands.useAccessGroups`, `channels.discord.dm.allowFrom`, `channels.discord.guilds`, `channels.discord.mediaMaxMb`.

Signal

Requires `signal-cli` installed separately. Configure under `channels.signal`.

BlueBubbles (iMessage)

Recommended iMessage integration. Requires a BlueBubbles server running on macOS.

Configure: `channels.bluebubbles.serverUrl`, `channels.bluebubbles.password`, `channels.bluebubbles.webhookPath`.

The BlueBubbles server runs on macOS; the OpenClaw Gateway can run on macOS or elsewhere.

iMessage (Legacy)

macOS-only. Uses the `imsg` integration (Messages must be signed in).

- `channels.imessage.groups` — when set, acts as a group allowlist. Include `"*"` to allow all.

Microsoft Teams

Configure a Teams app + Bot Framework, then add a `msteams` config section.

- `msteams.allowFrom` — user allowlist.
- `msteams.groupAllowFrom` or `msteams.groupPolicy: "open"` — group access.

WebChat

Uses the Gateway WebSocket directly. No separate port or config section needed.

Verify channel connectivity: Restart the gateway after channel configuration changes, then send a test message through your chosen channel.

```
openclaw gateway restart
openclaw status
```

3.5 Optional Apps

The Gateway alone delivers a complete experience. All apps below are optional and add extra features.

macOS App (OpenClaw.app):

- Menu bar control for Gateway health
- Voice wake + push-to-talk overlay

- WebChat + debug tools

- Remote gateway control over SSH

- Note: Signed builds required for macOS permissions to persist across rebuilds

iOS Node:

- Pairs as a node via the Bridge

- Voice trigger forwarding + Canvas surface

- Controlled via `openclaw nodes ...`

Android Node:

- Pairs via the same Bridge + pairing flow as iOS

- Exposes Canvas, Camera, and Screen capture commands

---

4. Daily Workflow

4.1 Message Flow

```
User sends message via channel (Telegram, WhatsApp, etc.)
        |
Gateway receives the message
        |
Gateway routes to the correct session (based on config)
        |
Session loads conversation history
        |
AI generates response (with full context + tool access)
        |
Response sent back through the channel
        |
History updated and saved to disk
```

The agent has access to tools configured in its profile: file system read/write, shell commands, web browsing, API calls, and any installed skills.

4.2 Sessions

A **session** is a persistent conversation with context. Every session has:

- A **session key** (unique identifier, e.g., `agent:main:main`)
- **Conversation history** (stored as JSONL files on disk)
- A **model** (which LLM to use)
- **Tools** (what the agent can access)

Sessions are independent. Each has its own history and context. Session types:

- **Main sessions:** Long-running, interactive (e.g., your primary chat with the agent).
- **Isolated sessions:** One-shot, typically for cron jobs. Wake up, do the task, terminate.

4.3 Chat Commands

Send these in any connected channel (WhatsApp, Telegram, Slack, Discord, Google Chat, Microsoft Teams, WebChat). Group-specific commands are owner-only.

Command	Description
`/status`	Compact session status (model, tokens, cost when available)
`/new` or `/reset`	Reset the session (clear context)
`/compact`	Compact session context (summarize to save tokens)
`/think `	Set thinking level: `off`, `minimal`, `low`, `medium`, `high`, `xhigh` (model-dependent)
`/verbose on|off`	Toggle verbose output
`/usage off|tokens|full`	Per-response usage footer
`/restart`	Restart the gateway (owner-only in groups)
`/activation mention|always`	Group activation toggle (groups only)

4.4 CLI Commands

Run from the terminal on the machine hosting the gateway.

```
openclaw status                   # Show gateway and session status
openclaw new                      # Create a new session
openclaw reset                    # Reset the current session
openclaw compact                  # Compact session context
openclaw think <level>            # Set thinking level
openclaw verbose on|off           # Toggle verbose mode
openclaw usage off|tokens|full    # Set usage display
openclaw restart                  # Restart the gateway
openclaw activation mention|always # Set activation mode
```

4.5 The Workspace

The workspace is the agent's persistent storage. Default root: `~/.openclaw/workspace` (configurable via `agents.defaults.workspace`).

```
~/.openclaw/workspace/
  |-- AGENTS.md             # Instructions for agents (operating manual)
  |-- SOUL.md               # Agent personality definition
  |-- TOOLS.md              # Tool documentation injected into context
  |-- skills/
  |   |-- <skill-name>/
  |       |-- SKILL.md      # Skill definition
  |-- memory/
  |   |-- WORKING.md        # Current task state
  |   |-- MEMORY.md         # Long-term curated knowledge
  |   |-- YYYY-MM-DD.md     # Daily notes
  |-- scripts/              # Utilities agents can run
  |-- config/               # Credentials, settings
```

Injected prompt files: `AGENTS.md`, `SOUL.md`, and `TOOLS.md` are automatically injected into the agent's context on session start.

4.6 Memory Management

AI sessions start fresh by default. Memory persistence requires writing to files.

The golden rule: If you want to remember something, write it to a file. "Mental notes" do not survive session restarts. Only files persist.

Memory stack:

1. **Session memory (built-in):** Conversation history stored as JSONL files. Agents can search their own past conversations.

2. **Working memory (`memory/WORKING.md`):** Current task state. Updated constantly. The most important file. When an agent wakes up, it reads `WORKING.md` first.

```
# WORKING.md

## Current Task
Researching competitor pricing for comparison page

## Status
Gathered reviews, need to verify credit calculations

## Next Steps
1. Test competitor free tier
```

```
2. Document findings
3. Post findings to task thread
```

3. **Daily notes (`memory/YYYY-MM-DD.md`):** Raw logs of what happened each day.

```
# 2026-01-31

## 09:15 UTC
- Posted research findings to comparison task
- Added competitive pricing data
- Moving to draft stage

## 14:30 UTC
- Reviewed first draft
- Suggested changes to pricing section
```

4. **Long-term memory (`memory/MEMORY.md`):** Curated important facts, key decisions, lessons learned, stable information.

When instructing an agent to "remember" something, ensure it updates a file rather than simply acknowledging the instruction.

---

5. Common Tasks

5.1 Add a Skill

Skills extend agent capabilities. They live in the workspace under `skills//SKILL.md`.

ClawHub (Skills Registry): OpenClaw includes a minimal skill registry. With ClawHub enabled, the agent can search for skills automatically and pull in new ones as needed. Browse available skills at the ClawHub skills URL.

Manual skill installation: Create a directory under `~/.openclaw/workspace/skills//` and add a `SKILL.md` file defining the skill's capabilities, instructions, and any required configuration.

5.2 Add an Agent

Agents are defined in `agents.list[]` in `openclaw.json`. Each agent entry has an `id`, `model`, `tools`, and optional `sandbox` configuration.

Each agent is a session with a specialized configuration. Define:

1. A unique session key (e.g., `agent:seo-analyst:main`)

2. A `SOUL.md` file defining the agent's identity and role

3. Tool access and sandbox configuration

4. Optional cron schedule for periodic wakeups

Example `SOUL.md`:

```
# SOUL.md -- Who You Are

**Name:** Analyst
**Role:** SEO Analyst

## Personality
Data-driven. Thinks in keywords and search intent.

## What You're Good At
- Keyword research and intent mapping
- Content optimization for search
- Competitor SERP analysis

## What You Care About
- Search visibility over vanity metrics
- Evidence-based decisions
```

Best practice: Start with 2-3 agents, not 10. Get them solid before adding more.

5.3 Schedule Cron Jobs

OpenClaw has a built-in cron system for scheduled agent wakeups.

```
# One-time daily task
openclaw cron add \
  --name "morning-check" \
  --cron "30 7 * * *" \
  --message "Check today's calendar and send me a summary"


# Recurring heartbeat (every 15 minutes, isolated session)
openclaw cron add \
  --name "agent-heartbeat" \
  --cron "*/15 * * * *" \
  --session "isolated" \
  --message "Check for work. If nothing, reply HEARTBEAT_OK."
```

When a cron fires:

1. The gateway creates or wakes a session

2. Sends the specified message to the AI

3. The AI responds (can use tools, send messages, etc.)

4. If `--session "isolated"`, the session terminates after completion

5.4 Send Messages Between Sessions

```
openclaw sessions send \
  --session "agent:seo-analyst:main" \
  --message "Can you review the latest keyword report?"
```

Sessions can: `list`, `history`, `send`, `spawn`. The `send` command messages another session with optional reply-back and announcement behavior (`REPLY_SKIP`, `ANNOUNCE_SKIP`).

5.5 Install a Local Model

```
openclaw models install llama3
openclaw agent set-model llama3
```

Important performance note: Local model inference requires significant hardware. Underpowered machines may produce extremely slow token generation. API-backed models (Anthropic, OpenAI, Google) provide dramatically faster responses and are recommended for production use.

5.6 Enable Browser Control

Add to `openclaw.json`:

```
{
  "browser": {
    "enabled": true,
    "color": "#FF4500"
  }
}
```

The browser tool allows the agent to control a browser instance. When sandboxing is enabled, the browser can optionally run inside the sandbox (see `agents.defaults.sandbox.browser`).

5.7 View Logs & Health

```
openclaw status              # Quick status check
openclaw gateway status      # Gateway-specific status and binding info
```

OpenClaw provides health checks, logging, and gateway lock diagnostics. Refer to the official docs for:

- Health checks

- Gateway lock

- Background process monitoring

- Logging configuration

- Browser troubleshooting (Linux)

5.8 Upgrade OpenClaw

```
npm update -g openclaw
openclaw gateway restart
```

Verify:

```
openclaw status
```

---

6. Troubleshooting

Symptom: Gateway won't start

| Cause | Fix |

|---|---|

| Port 18789 already in use | Check for existing process: `lsof -i :18789` (macOS/Linux) or `netstat -ano \| findstr 18789` (Windows). Kill the conflicting process or change the port in `openclaw.json`. |

| Invalid JSON in config | Validate `~/.openclaw/openclaw.json` syntax. Common issues: missing commas, trailing commas (depending on parser), unquoted keys. |

| Missing API key | Ensure your provider's API key is set as an environment variable and your shell profile is sourced. |

Symptom: Agent doesn't respond in channel

| Cause | Fix |

|---|---|

| Channel not configured | Verify channel config in `openclaw.json`. Restart gateway after changes. |

| Bot token invalid/expired | Regenerate the bot token from the platform (Telegram BotFather, Discord Developer Portal, etc.) and update config. |

| Allowlist blocking | Check `allowFrom` / `groups` settings. If the allowlist is set and doesn't include the sender, messages are silently dropped. |

| Activation mode set to `mention` | In groups, if `activation` is `mention`, the agent only responds when @mentioned. Switch to `always` or @mention the bot. |

Symptom: Agent forgets context between messages

| Cause | Fix |

|---|---|

| Session reset | Check if `/new` or `/reset` was triggered. |

| Context window overflow | Use `/compact` to summarize and compress context. |

| Isolated session used | Isolated sessions (from cron) don't carry forward. For persistent conversations, use main sessions. |

| Not writing to files | Agents must write important state to `WORKING.md` or other files. "Mental notes" are lost on session restart. |

Symptom: Sandbox blocks a tool unexpectedly

```
openclaw sandbox explain
```

This command inspects the effective sandbox mode, tool policy, and relevant config keys. Common issues:

| Cause | Fix |

|---|---|

| Tool denied by policy | Tool allow/deny policies apply before sandbox rules. Check `agents.defaults.tools` and per-agent overrides. |

| `setupCommand` fails | Default sandbox network is `"none"` (no egress). Package installs require `agents.defaults.sandbox.docker.network` to be set. Also check `readOnlyRoot` and `user` settings. |

| Skill needs Node in sandbox | Default sandbox image does not include Node. Bake a custom image or use `sandbox.docker.setupCommand`. |

Symptom: High API costs

| Cause | Fix |

|---|---|

| Expensive model on heartbeats | Use cheaper models for routine/heartbeat tasks. Reserve expensive models for creative work. See [Section 8.5](#85-cost-control). |

| Heartbeat interval too frequent | 15 minutes is the recommended balance. Every 5 minutes is expensive with minimal benefit. |

| Too many agents | Scale back to 2-3 agents. Add more only after the initial set is tuned. |

Symptom: Gateway exposed to the internet

See [Section 7.1](#71-5-minute-security-check) immediately. Run:

```
openclaw gateway status
# If bind is NOT loopback, fix immediately
```

---

7. Security Hardening

Why this matters: The gateway was designed for local use. When run on a server or VPS without hardening, it exposes conversation histories, API keys, OAuth tokens, and shell access to anyone who can reach the port. Researchers have found thousands of exposed instances via network scanning tools.

Known attack vector: Prompt injection via email. A crafted email sent to an agent with email integration can instruct the AI to read and forward private messages. This is not a bug in OpenClaw; it is inherent to AI agents that both read external content and take actions.

7.1 5-Minute Security Check

Check 1: What address is the gateway listening on?

```
openclaw gateway status
```

- `bind=loopback (127.0.0.1)` — Only the local machine can connect. **Good.**
- `bind=lan` or `bind=0.0.0.0` — Listening on all interfaces. **Fix immediately.**

What this means:

- `127.0.0.1` (loopback) = Only your machine can connect. Soundproof room.
- `0.0.0.0` (all interfaces) = Anyone on your network (or the internet, if ports are open) can connect. Megaphone.

Check 2: Can you access it externally?

If running on a remote server, from a separate device (not on the same network):

```
nc -zv <SERVER-IP> 18789
```

If you get a connection, or if browsing to `http://:18789` shows the interface without a password, you are exposed.

7.2 15-Minute Lockdown

Complete these steps in order.

Step 1: Bind to Localhost Only (2 min)

Edit `~/.openclaw/openclaw.json`:

```
{
  "gateway": {
    "bind": "loopback",
    "port": 18789
  }
}
```

```
openclaw gateway restart
```

Verify:

```
openclaw gateway status
# Confirm: bind=loopback (127.0.0.1)
```

Step 2: Lock Down File Permissions (2 min)

macOS/Linux:

```
chmod 700 ~/.openclaw
chmod 600 ~/.openclaw/openclaw.json
chmod 700 ~/.openclaw/credentials
```

What the permission numbers mean:

- `700` = Only the owner can access the directory

- `600` = Only the owner can read/write the file

Or let OpenClaw fix it automatically:

```
openclaw security audit --fix
```

Step 3: Disable Network Broadcasting (1 min)

OpenClaw uses mDNS (Bonjour) for device discovery by default. This can expose information on your local network.

Add to your shell profile (`~/.bashrc` or `~/.zshrc`):

```
export CLAWDBOT_DISABLE_BONJOUR=1
```

Reload:

```
source ~/.zshrc  # or source ~/.bashrc
```

Step 4: Run the Security Audit (2 min)

```
openclaw security audit --deep
```

Checks file permissions, network exposure, configuration issues, and known vulnerabilities.

To auto-fix:

```
openclaw security audit --deep --fix
```

Review the output. It reports what was found and what was changed.

Step 5: Set Up Authentication (3 min)

Option A: Token authentication

```
export CLAWDBOT_GATEWAY_TOKEN="$(openssl rand -hex 32)"
```

Option B: Password authentication

Add to `openclaw.json`:

```
{
  "gateway": {
    "auth": {
      "mode": "password"
    }
  }
}
```

```
export CLAWDBOT_GATEWAY_PASSWORD="your-secure-password-here"
```

Restart the gateway after authentication changes.

Step 6: Ensure Node.js Is Current (3 min)

```
node --version
# Must be >= 22.12.0
```

Upgrade if needed (see [Section 2.2](#22-install-nodejs)).

7.3 Sandboxing

OpenClaw can run tool execution inside Docker containers to limit blast radius. This is optional but **strongly recommended** for group/channel sessions where other users can trigger agent actions.

What gets sandboxed:

- Tool execution (exec, read, write, edit, apply_patch, process, etc.)

- Optional sandboxed browser

What is NOT sandboxed:

- The Gateway process itself

- Tools explicitly allowed to run on the host via `tools.elevated`

- Elevated exec (runs on the host, bypasses sandboxing)

Sandbox Modes

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

| Mode | Behavior |

|---|---|

| `"off"` | No sandboxing. All tools run on the host. |

| `"non-main"` | Sandbox only non-main sessions (groups, channels). Main session runs on host. **Recommended starting point.** |

| `"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 and count as non-main.

Sandbox Scope

`agents.defaults.sandbox.scope` controls container granularity:

Scope	Behavior
`"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:

Access	Behavior
`"none"` (default)	Tools see a sandbox workspace under `~/.openclaw/sandboxes`
`"ro"`	Mounts agent workspace read-only at `/agent` (disables write/edit/apply_patch)
`"rw"`	Mounts agent workspace read/write at `/workspace`

Minimal Sandbox Enable

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

Sandbox Images

Build the default sandbox image:

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

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

Build the sandboxed browser image:

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

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

Custom Bind Mounts

`agents.defaults.sandbox.docker.binds` mounts additional host directories into the container.

Format: `host:container:mode`

```
{
  "agents": {
    "defaults": {
      "sandbox": {
        "docker": {
          "binds": [
            "/path/to/source:/source:ro",
            "/var/run/docker.sock:/var/run/docker.sock"
          ]
        }
      }
    }
  }
}
```

Security notes on binds:

- Binds bypass the sandbox filesystem. They expose host paths with whatever mode you set (`:ro` or `:rw`).

- Sensitive mounts (docker.sock, secrets, SSH keys) should be `:ro` unless absolutely required.

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

setupCommand (One-Time Container Setup)

`setupCommand` runs once after the sandbox container is created (not on every run). 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) — package installs will fail without network.

- `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` for skill API keys.

Tool Policy + Escape Hatches

- Tool allow/deny policies apply **before** sandbox rules. If a tool is denied globally or per-agent, sandboxing does not bring it back.

- `tools.elevated` is an explicit escape hatch that runs exec on the host.

- To hard-disable elevated exec, use tool policy deny.

Sandbox default allowlist: bash, process, read, write, edit, sessions_list, sessions_history, sessions_send, sessions_spawn.

Sandbox default denylist: browser, canvas, nodes, cron, discord, gateway.

Debugging sandbox issues:

```
openclaw sandbox explain
```

Multi-Agent Sandbox Overrides

Each agent can override sandbox + tools: `agents.list[].sandbox` and `agents.list[].tools` (plus `agents.list[].tools.sandbox.tools` for sandbox tool policy).

7.4 Remote Access (Tailscale)

For accessing OpenClaw on a remote server without exposing the gateway to the public internet.

Install Tailscale on the server:

```
curl -fsSL https://tailscale.com/install.sh | sh
sudo tailscale up
```

Install Tailscale on your personal device and log in with the same account.

Configure OpenClaw to use Tailscale:

```
{
```

```
  "gateway": {
    "bind": "loopback",
    "tailscale": {
      "mode": "serve"
    }
  }
}
```

Access your OpenClaw instance securely from any device on your Tailscale network.

7.5 Server Hardening (VPS/Cloud)

If running on AWS, DigitalOcean, Hetzner, or similar providers, apply these additional protections.

SSH Hardening

Disable password authentication (use SSH keys instead):

Edit `/etc/ssh/sshd_config`:

```
PasswordAuthentication no
PermitRootLogin no


sudo systemctl restart sshd
```

Firewall (UFW)

UFW blocks all incoming traffic except what you explicitly allow.

```
# Check if installed
sudo ufw status

# Install if needed (Ubuntu/Debian)
sudo apt update && sudo apt install ufw -y

# Set defaults: deny incoming, allow outgoing
sudo ufw default deny incoming
sudo ufw default allow outgoing

# CRITICAL: Allow SSH before enabling (or you'll lock yourself out)
sudo ufw allow ssh

# If using non-standard SSH port:
sudo ufw allow 2222/tcp  # Replace with your port

# If using Tailscale:
```

```
sudo ufw allow in on tailscale0

# Enable the firewall
sudo ufw enable

# Verify
sudo ufw status verbose
```

Expected output:

```
Status: active
Default: deny (incoming), allow (outgoing), disabled (routed)

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW IN    Anywhere
Anywhere on tailscale0     ALLOW IN    Anywhere
```

Do NOT allow port 18789 through UFW. Access the gateway only through localhost or Tailscale.

7.6 Threat Model Summary

| Threat | Mitigation |

|---|---|

| Gateway exposed to internet | Bind to loopback. Use Tailscale for remote access. Firewall port 18789. |

| API key theft from config | `chmod 600` on config files. Run `openclaw security audit --fix`. |

| Prompt injection via email/message | Enable sandboxing for non-main sessions. Use models with strong prompt-injection defenses. Avoid adding the bot to public group chats. |

| mDNS/Bonjour discovery | Set `CLAWDBOT_DISABLE_BONJOUR=1`. |

| Agent accesses host filesystem | Enable sandbox with `workspaceAccess: "none"`. Review tool policies. |

| Compromised channel triggers shell access | Sandbox non-main sessions. Deny elevated exec in tool policy for channel sessions. |

Minimum safe config checklist:

1. `gateway.bind: "loopback"`

2. Gateway authentication enabled (token or password)

3. `agents.defaults.sandbox.mode: "non-main"` (at minimum)

4. File permissions locked (`chmod 700`/`600`)

5. mDNS broadcasting disabled

6. `openclaw security audit --deep` passes clean

7. Node.js >= 22.12.0

---

8. Power Users: Optimization & Advanced Patterns

8.1 Multi-Agent Architecture

OpenClaw sessions are independent. Each can have its own personality (via `SOUL.md`), memory files, cron schedule, and tool access. A multi-agent system is simply multiple sessions, each with specialized configuration.

Session key convention:

```
agent:main:main               --> Coordinator / Squad Lead
agent:product-analyst:main    --> Product Analyst
agent:customer-researcher:main --> Customer Researcher
agent:seo-analyst:main        --> SEO Analyst
agent:content-writer:main     --> Content Writer
agent:social-media:main       --> Social Media Manager
agent:designer:main           --> Designer
agent:email-marketing:main    --> Email Marketing Specialist
agent:developer:main          --> Developer
agent:docs:main               --> Documentation Specialist
```

When you send a message to a specific session, only that agent receives it. Histories are separate.

Agent levels (recommended access tiers):

- **Intern:** Needs approval for most actions. Learning the system.

- **Specialist:** Works independently in their domain.

- **Lead:** Full autonomy. Can make decisions and delegate.

Scaling advice:

- Start with 1 coordinator + 1 specialist. Get them solid before adding more.

- Each agent needs a distinct `SOUL.md` that constrains their role. An agent "good at everything" is mediocre at everything.

- The `AGENTS.md` file is the shared operating manual. Every agent reads it on startup. It covers file locations, memory conventions, tool access, when to speak vs. stay quiet, and how to use shared infrastructure.

8.2 The Heartbeat System

Problem: Always-on agents burn API credits doing nothing. Always-off agents can't respond to work.

Solution: Scheduled heartbeats via cron. Each agent wakes every 15 minutes in an isolated session.

```
openclaw cron add \
  --name "analyst-heartbeat" \
  --cron "0,15,30,45 * * * *" \
  --session "isolated" \
  --message "You are the Product Analyst. Check for new tasks assigned to you, check for @mentions,
```

Stagger the schedule so agents do not all wake at once:

```
:00  Email Marketing
:02  Product Analyst
:04  Developer
:06  Content Writer
:07  Designer
:08  SEO Analyst
:10  Customer Researcher
:12  Social Media
```

What happens during a heartbeat:

1. **Load context:** Read `WORKING.md`. Read recent daily notes. Check session memory if needed.
2. **Check for urgent items:** Am I @mentioned anywhere? Are there tasks assigned to me?
3. **Scan activity feed:** Any discussions I should contribute to? Any decisions that affect my work?
4. **Take action or stand down:** If there's work, do it. If nothing, report `HEARTBEAT_OK`.

The `HEARTBEAT.md` file tells agents what to check:

```
# HEARTBEAT.md

## On Wake
- [ ] Check memory/WORKING.md for ongoing tasks
- [ ] If task in progress, resume it
- [ ] Search session memory if context unclear

## Periodic Checks
- [ ] Check shared task system for @mentions
- [ ] Check assigned tasks
- [ ] Scan activity feed for relevant discussions
```

Why 15 minutes?

- Every 5 minutes: Too expensive. Agents wake too often with nothing to do.
- Every 30 minutes: Too slow. Work sits waiting.
- Every 15 minutes: Good balance. Most work gets attention quickly without excessive costs.

8.3 Inter-Agent Communication

Option 1: Direct session messaging

```
openclaw sessions send \
  --session "agent:seo-analyst:main" \
  --message "Can you review the latest keyword report?"
```

The coordinator agent can send messages directly to any specialist's session.

Option 2: Shared database

All agents read and write to a shared database (e.g., Convex, Notion, or even a shared JSON file). When one agent posts a comment, everyone can see it on their next heartbeat.

A shared database approach creates a record of all communication and makes task history auditable.

Recommended data schema (if building a shared task system):

- **Agents table:** name, role, status (idle/active/blocked), current task, session key
- **Tasks table:** title, description, status (inbox/assigned/in_progress/review/done), assignees
- **Messages table:** task reference, sender, content, attachments
- **Activities table:** event type, agent, message (activity feed)
- **Documents table:** title, content (markdown), type (deliverable/research/protocol), task reference
- **Notifications table:** mentioned agent, content, delivered status

Agents interact with the shared system via CLI commands or API calls from within their sessions.

8.4 Task Lifecycle & Coordination

Task statuses:

```
Inbox --> Assigned --> In Progress --> Review --> Done
                          |
                       Blocked (needs something resolved)
```

Example flow:

1. **Day 1:** Create a task and assign it to the SEO Analyst and Content Writer. SEO Analyst posts keyword research.

2. **Day 1-2:** Customer Researcher sees it in the activity feed and adds competitor intel. Product Analyst tests both products and documents UX differences.

3. **Day 2:** Content Writer starts drafting, using all research.

4. **Day 3:** Content Writer posts first draft. Status moves to Review. You review, give feedback. Writer revises. Done.

All comments on a single task. Full history preserved. Anyone can see the whole journey.

8.5 Cost Control

API costs are the primary ongoing expense. Key strategies:

| Strategy | Implementation |

|---|---|

| Cheap models for heartbeats | Heartbeat tasks (checking for work, reporting HEARTBEAT_OK) do not need the most capable model. Use a cheaper model for routine checks. Reserve expensive models for creative/analytical work. |

| Isolated sessions for cron | Use `--session "isolated"` for cron jobs. The session wakes, does its task, and terminates. No lingering context. |

| Stagger heartbeats | Prevent multiple agents from invoking API calls simultaneously. Stagger by 2-minute intervals. |

| Use `/compact` | When conversation context grows large, compact it to reduce token usage per request. |

| Start with fewer agents | More agents = more heartbeats = more cost. Scale up only when the existing agents are well-tuned. |

| Monitor with `/usage full` | Enable per-response usage footers to track token consumption in real time. |

Model cost hierarchy (approximate, from most to least expensive):

- Anthropic Claude Opus > OpenAI GPT-5 > Anthropic Claude Sonnet > Google Gemini > Smaller/local models

Switching heartbeat and routine tasks to Google Gemini models or similar can reduce costs dramatically without sacrificing output quality on simple check-in tasks.

8.6 Multi-Model Workflows

OpenClaw supports using different models for different agents or tasks.

Configuration: Set the model per-agent in `agents.list[]`:

```
{
  "agents": {
    "list": [
      {
        "id": "coordinator",
        "model": "anthropic/claude-opus-4-6"
      },
      {
        "id": "heartbeat-checker",
        "model": "google/gemini-3-flash"
      }
    ]
  }
}
```

Strategy:

- **Orchestration/creative work:** Use the most capable model available (e.g., Claude Opus).

- **Routine checks/heartbeats:** Use a fast, cheap model (e.g., Gemini Flash).

- **Code generation/review:** Match the model to the task complexity.

The orchestrator (OpenClaw) decides what happens when, which tools are used, and which skills are needed. The underlying model can be swapped without changing the agent's behavior or personality.

Local models: OpenClaw can use locally-run models as well (via `openclaw models install`). This eliminates API costs entirely but requires capable hardware and yields slower responses.

8.7 Automation Loops

OpenClaw agents can run continuously on autopilot via cron-driven loops.

Common automation patterns:

| Pattern | Implementation |

|---|---|

| **Morning briefing** | Cron at 07:30 sends "Check calendar, weather, and overnight task results. Send summary to Telegram." |

| **Competitor monitoring** | Cron every 4 hours sends "Search for new content from competitors. Flag outliers. Post to task board." |

| **Email triage** | Connect email integration. Agent reads incoming mail, categorizes, drafts responses, flags urgent items. |

| **Content pipeline** | Agent chain: Researcher gathers data -> Writer drafts -> Reviewer provides feedback -> Writer revises. |

| **Ad performance monitoring** | Daily cron: "Check ad spend, CTR, ROAS. Pause underperformers. Alert on anomalies." |

| **Landing page health** | Cron every 2 hours: "Ping key URLs. Alert if down or slow." |

Proactive behavior: Agents can be instructed to be proactive. Instead of only responding to direct requests, they can propose daily action plans, flag issues they discover, and initiate work based on patterns they observe.

Include in the agent's `SOUL.md` or `AGENTS.md`:

```
## Proactive Behavior
- Come to me every day with an action plan
- If you spot an issue, flag it immediately
- If a task could benefit from your expertise, contribute without being asked
```

8.8 Notification System

@Mentions: Reference another agent by name in a comment (e.g., "@SEO-Analyst") and they receive a notification on their next heartbeat.

Delivery mechanism: A daemon process (e.g., running via pm2) polls the shared task system every few seconds for undelivered notifications. When found, it sends the notification to the target agent's session via `openclaw sessions send`. If the agent is asleep, the notification stays queued until their next heartbeat activates the session.

Thread subscriptions: When an agent interacts with a task (comments, gets @mentioned, gets assigned), they are automatically subscribed to all future comments on that task. No @mention needed for subsequent messages. This makes conversations flow naturally.

8.9 Daily Standup Automation

Schedule a daily cron that compiles agent activity into a summary and sends it to your primary channel.

```
openclaw cron add \
  --name "daily-standup" \
  --cron "0 23 * * *" \
  --session "isolated" \
  --message "Compile today's standup. List: completed tasks, in-progress work, blocked items, items
```

Expected output format:

```
DAILY STANDUP -- Jan 30, 2026

COMPLETED TODAY
- Content Writer: Blog post (2,100 words)
- Social Media: 10 tweets drafted for approval
- Customer Researcher: Research for comparison pages

IN PROGRESS
- SEO Analyst: Strategy for integration pages
- Email Marketing: Trial onboarding sequence (3/5 emails)

BLOCKED
- Designer: Waiting for brand colors for infographic

NEEDS REVIEW
- Blog post draft
- Trial email sequence

KEY DECISIONS
- Lead with pricing transparency in comparisons
- Deprioritized low-volume comparison page
```

This provides daily visibility without constantly watching the task board.

8.10 Performance Tuning

| Area | Recommendation |

|---|---|

| **Heartbeat frequency** | 15 minutes is the sweet spot. Adjust per agent if some need faster response. |

| **Context management** | Use `/compact` regularly. Keep `WORKING.md` concise. Archive old daily notes. |

| **Parallel agents** | Stagger cron jobs by 2+ minutes to avoid API rate limits and cost spikes. |

| **Model selection** | Match model capability to task. Do not use the most expensive model for simple checks. |

| **Memory discipline** | Enforce file-based memory. Review agent outputs for "I'll remember that" without file writes. |

| **Session cleanup** | Use isolated sessions for one-shot tasks. Don't let stale sessions accumulate context. |

| **Sandbox overhead** | `scope: "agent"` or `scope: "shared"` reduces container creation overhead vs. `scope: "session"`. Trade-off is isolation granularity. |

| **Hardware (local models)** | If running local models, ensure adequate RAM and GPU. Underpowered hardware leads to extremely slow token generation (potentially 40+ minutes for a few thousand tokens). |

---

9. Appendix: Key Commands & Config Glossary

9.1 CLI Command Reference

Command	Description
`openclaw init`	Initialize OpenClaw (creates `~/.openclaw/` and config scaffold)
`openclaw gateway start`	Start the gateway daemon
`openclaw gateway restart`	Restart the gateway daemon
`openclaw gateway status`	Show gateway status (bind address, port, health)
`openclaw status`	Show overall status
`openclaw new`	Create a new session
`openclaw reset`	Reset the current session
`openclaw compact`	Compact session context (summarize)
`openclaw think `	Set thinking level (`off|minimal|low|medium|high|xhigh`)
`openclaw verbose on|off`	Toggle verbose output
`openclaw usage off|tokens|full`	Set usage display mode
`openclaw restart`	Restart the gateway
`openclaw activation mention|always`	Set group activation mode
`openclaw models install `	Install a local model
`openclaw agent set-model `	Set the agent's model
`openclaw cron add --name "..." --cron "..." --message "..."`	Add a scheduled cron job
`openclaw cron add ... --session "isolated"`	Add a cron job with isolated session
`openclaw sessions send --session "..." --message "..."`	Send a message to a specific session
`openclaw sandbox explain`	Inspect effective sandbox mode and tool policy
`openclaw security audit`	Run basic security audit
`openclaw security audit --fix`	Run audit and auto-fix issues
`openclaw security audit --deep`	Run deep security audit
`openclaw security audit --deep --fix`	Deep audit with auto-fix
`pnpm openclaw channels login`	Link WhatsApp device (stores creds in `~/.openclaw/credentials`)

9.2 Chat Command Reference

Send via any connected channel (WhatsApp, Telegram, Slack, Discord, Google Chat, Microsoft Teams, WebChat).

Command	Description
`/status`	Compact session status (model, tokens, cost)
`/new`	Reset the session
`/reset`	Reset the session
`/compact`	Compact session context
`/think `	`off|minimal|low|medium|high|xhigh`
`/verbose on|off`	Toggle verbose output
`/usage off|tokens|full`	Per-response usage footer
`/restart`	Restart gateway (owner-only in groups)
`/activation mention|always`	Group activation toggle (groups only)

9.3 Configuration Keys

All keys are in `~/.openclaw/openclaw.json`.

Key	Type	Description
`agent.model`	string	Model identifier (e.g., `"anthropic/claude-opus-4-6"`)
`gateway.bind`	string	Network binding: `"loopback"`, `"lan"`, or IP address
`gateway.port`	number	Gateway port (default: `18789`)
`gateway.auth.mode`	string	Authentication mode: `"password"`
`gateway.tailscale.mode`	string	Tailscale integration mode: `"serve"`
`agents.defaults.workspace`	string	Workspace root path (default: `~/.openclaw/workspace`)
`agents.defaults.sandbox`	object	Default sandbox configuration (see 9.4)
`agents.defaults.tools`	object	Default tool allow/deny policy
`agents.list[]`	array	Array of agent definitions
`agents.list[].id`	string	Agent identifier
`agents.list[].model`	string	Model override for this agent
`agents.list[].tools`	object	Per-agent tool policy

`agents.list[].sandbox`	object	Per-agent sandbox overrides
`channels.telegram.botToken`	string	Telegram bot token
`channels.telegram.groups`	object	Group allowlist (`"*"` for all)
`channels.telegram.allowFrom`	array	User allowlist
`channels.telegram.webhookUrl`	string	Webhook URL
`channels.telegram.webhookSecret`	string	Webhook secret
`channels.whatsapp.allowFrom`	array	User allowlist
`channels.whatsapp.groups`	object	Group allowlist
`channels.slack.botToken`	string	Slack bot token
`channels.slack.appToken`	string	Slack app token
`channels.discord.token`	string	Discord bot token
`channels.discord.dm.allowFrom`	array	DM allowlist
`channels.discord.guilds`	object	Guild configuration
`channels.discord.mediaMaxMb`	number	Max media size in MB
`channels.signal`	object	Signal configuration (requires `signal-cli`)
`channels.bluebubbles.serverUrl`	string	BlueBubbles server URL
`channels.bluebubbles.password`	string	BlueBubbles password
`channels.bluebubbles.webhookPath`	string	Webhook path
`channels.imessage.groups`	object	iMessage group allowlist (legacy)
`msteams.allowFrom`	array	Teams user allowlist
`msteams.groupAllowFrom`	array	Teams group allowlist
`msteams.groupPolicy`	string	`"open"` for open group access
`browser.enabled`	boolean	Enable browser control
`browser.color`	string	Browser overlay color
`commands.native`	object	Native commands config (Discord)
`commands.text`	object	Text commands config (Discord)
`commands.useAccessGroups`	boolean	Use access groups (Discord)
`tools.elevated`	object	Elevated exec configuration (host escape hatch)

9.4 Sandbox Configuration Keys

Key	Type	Default	Description
`agents.defaults.sandbox.mode`	string	—	`"off"`, `"non-main"`, or `"all"`

| `agents.defaults.sandbox.scope` | string | `"session"` | `"session"`, `"agent"`, or `"shared"` |

| `agents.defaults.sandbox.workspaceAccess` | string | `"none"` | `"none"`, `"ro"`, or `"rw"` |

| `agents.defaults.sandbox.docker.binds` | array | `[]` | Host bind mounts (`"host:container:mode"`) |

| `agents.defaults.sandbox.docker.network` | string | `"none"` | Docker network mode |

| `agents.defaults.sandbox.docker.setupCommand` | string | — | One-time setup command (runs in container) |

| `agents.defaults.sandbox.docker.env` | object | — | Environment variables for sandbox |

| `agents.defaults.sandbox.docker.readOnlyRoot` | boolean | — | Read-only root filesystem |

| `agents.defaults.sandbox.docker.user` | string | — | Container user (e.g., `"0:0"` for root) |

| `agents.defaults.sandbox.browser` | object | — | Sandbox browser configuration |

| `agents.defaults.sandbox.browser.autoStart` | boolean | — | Auto-start browser in sandbox |

| `agents.defaults.sandbox.browser.autoStartTimeoutMs` | number | — | Auto-start timeout |

| `agents.defaults.sandbox.browser.allowHostControl` | boolean | — | Allow sandbox to target host browser |

9.5 Workspace File Reference

| File | Purpose |

|---|---|

| `AGENTS.md` | Operating manual for agents. Read on startup. Covers file locations, memory conventions, tool access, communication protocols. |

| `SOUL.md` | Agent personality and identity definition. Name, role, personality traits, strengths, priorities. |

| `TOOLS.md` | Tool documentation injected into agent context. |

| `HEARTBEAT.md` | Checklist for heartbeat wakeups. What to check, in what order. |

| `memory/WORKING.md` | Current task state. The most critical file. Read first on every wakeup. |

| `memory/MEMORY.md` | Long-term curated knowledge. Key decisions, lessons, stable facts. |

| `memory/YYYY-MM-DD.md` | Daily notes. Raw log of what happened each day. |

| `skills//SKILL.md` | Skill definition files. |

9.6 Environment Variables

| Variable | Description |

|---|---|

| `ANTHROPIC_API_KEY` | Anthropic API key |

| `OPENAI_API_KEY` | OpenAI API key |

| `GOOGLE_API_KEY` | Google AI API key |

| `TELEGRAM_BOT_TOKEN` | Telegram bot token (overrides config) |

| `SLACK_BOT_TOKEN` | Slack bot token (overrides config) |

| `SLACK_APP_TOKEN` | Slack app token (overrides config) |

| `DISCORD_BOT_TOKEN` | Discord bot token (overrides config) |

| `CLAWDBOT_GATEWAY_TOKEN` | Gateway authentication token |

| `CLAWDBOT_GATEWAY_PASSWORD` | Gateway authentication password |

| `CLAWDBOT_DISABLE_BONJOUR` | Set to `1` to disable mDNS/Bonjour broadcasting |

This guide covers OpenClaw from first install through multi-agent orchestration. For the full configuration reference, architecture deep dives, and platform-specific internals, consult the official documentation at docs.openclaw.ai.