OpenClaw Super Guide

Deploy + Operate: a complete, practical manual for installing OpenClaw and running reliable agent workflows

Updated: February 2026
Author: ZeroSignal

A deployment-focused install guide combined with an operator playbook for running AI agents safely, repeatably, and profitably.

License and disclaimer

This document is provided for educational purposes. You are responsible for complying with applicable laws, service terms, and security requirements. Do not run OpenClaw or any automation on systems you do not own or do not have explicit permission to administer.

OpenClaw and other product names may be trademarks of their respective owners. No affiliation is implied.

Table of contents

Part I - Deploy

This section is intentionally procedural. Follow it when you want OpenClaw running locally, in Docker, or on a server with predictable behavior.

1. What OpenClaw is (and what it is not)

OpenClaw is an agent runner. You define a model provider, permissions, a workspace, and agent prompts. OpenClaw then executes tasks by calling the model and (optionally) tools on your machine or inside a container.

It is not a magic remote desktop: treat it like automation code. Keep scopes small, use least-privilege permissions, and add verification steps for anything important.

2. Quick Start (15 minutes)

Goal: get a local gateway running, confirm it responds, and run a single test task.

Step A - Pick an install path

- macOS: install via package manager and run locally.
- Windows: either Docker (recommended for isolation) or WSL2 + local install.
- Docker: best for reproducibility and easy resets.
- VPS/Linux: best for always-on workloads.

Step B - Install OpenClaw

Use the official install command for your platform. If you already installed via npm/pnpm, skip to configuration.

```
# Example (Node-based install)
# Use the command recommended by your distribution of OpenClaw
npm install -g openclaw
```

Step C - Create or open your config

```
openclaw configure
# or
openclaw config open
```

Step D - Start the gateway

```
openclaw gateway start
# if using docker compose:
docker compose up -d openclaw-gateway
```

Step E - Verify

```
openclaw doctor
openclaw gateway status
# optional: view logs
openclaw gateway logs --tail=200
```

If doctor reports auth/provider issues, fix provider keys first (Section 4).

3. Install paths by platform

Use the smallest, simplest path that matches your environment. If you want the least amount of host pollution, choose Docker.

3.1 macOS (local install)

- Prereqs: Node.js LTS, a shell (zsh), and a provider key (OpenAI/OpenRouter/etc.).
- Install OpenClaw, run configure, then start the gateway as a background service if desired.

```
# Install
npm install -g openclaw

# Configure
openclaw configure

# Run
openclaw gateway start

# Verify
openclaw doctor
```

Persistence options: LaunchAgent (recommended) or a user-level service manager. Keep logs in a known directory and rotate them.

3.2 Windows

Two supported approaches: Docker Desktop (recommended) or WSL2 + local install. Docker is easiest to reset and isolate.

Option A - Docker Desktop + Compose

```
# In your OpenClaw project directory
docker compose up -d

docker compose ps

docker compose logs --tail=200 openclaw-gateway
```

Option B - WSL2 + local install

```
# Inside WSL2 Ubuntu
sudo apt update
# install Node.js LTS per NodeSource or distro packages
npm install -g openclaw
openclaw configure
openclaw gateway start
openclaw doctor
```

Windows note: prefer storing workspaces and datasets on a fast local SSD; avoid network shares for heavy I/O workflows.

3.3 Linux / VPS

- Choose: containerized (Compose) or local (systemd).
- Ensure firewall rules allow only what you need. Default: bind gateway to localhost and front it with a reverse proxy if necessary.
- Store secrets in environment variables or a secret manager.

```
# systemd-style approach (high level)
# 1) install openclaw
# 2) create /etc/openclaw/env with provider keys
# 3) create a systemd unit to run: openclaw gateway start
# 4) enable + start the service
```

4. Provider setup (keys, profiles, priority)

Your config needs (1) a provider credential and (2) a model selection policy (primary + fallbacks). Keep credentials out of git.

Recommended pattern

- Primary: one model you trust for most work.
- Worker: a cheaper model for bulk tasks.
- Escalation: a stronger model for complex reasoning/coding.
- Fallbacks: a safe default if a provider errors.

```
# Example environment variables (do not commit)
OPENAI_API_KEY=...
OPENROUTER_API_KEY=...
ANTHROPIC_API_KEY=...
```

After setting keys, re-run:

```
openclaw doctor --fix
openclaw gateway restart
openclaw gateway status
```

If you see authentication header errors, confirm the gateway process actually has the env vars (common issue when running as a service).

5. Persistence and service management

If you want OpenClaw to survive reboots, run it under a service manager appropriate for your platform.

- macOS: LaunchAgent (user) or LaunchDaemon (system).
- Linux: systemd unit.
- Windows: Docker Desktop auto-start, or a Scheduled Task that starts your compose stack.

Service hardening

- Run as a non-admin user.
- Restrict file system permissions to a workspace directory.
- Bind network listeners to localhost unless you truly need LAN/WAN access.
- Rotate logs; cap disk usage.

6. Verification and smoke tests

Do these every time you update OpenClaw, change provider settings, or change your agent tool permissions.

- Config validates (no schema errors).
- Gateway starts and stays up for 5+ minutes.
- One model call succeeds (simple prompt).
- Workspace write succeeds (creates a file).
- Tool calls behave as expected (only permitted tools work).

```
# Minimal smoke test task
openclaw run "Write a file named /shared/smoke_test.txt containing the text 'ok'"
```

If running inside Docker, ensure /shared is a bind mount to a host directory you can inspect.

7. Troubleshooting (top failure modes)

Rate limit / quota

- Confirm you are using the intended provider key.
- Lower concurrency; retry with exponential backoff.
- Switch worker tasks to a cheaper model; reserve strong models for escalation.

401 missing authentication header

- Confirm the env var is set in the same context as the gateway (service vs interactive shell).
- In Docker: confirm the variable exists in the container: printenv | grep -i openrouter.
- Restart gateway after changing env vars.

```
# Docker check
docker compose exec openclaw-gateway sh -lc "printenv | sort | egrep -i
'OPENAI|OPENROUTER|ANTHROPIC'"
```

Gateway keeps restarting

- Read the last 200 lines of logs; find the first fatal error.
- Validate config JSON and remove trailing commas / invalid fields.
- Check bind mounts exist and are writable.

```
docker compose logs --tail=200 openclaw-gateway
```

Part II - Operate

This section turns OpenClaw from 'installed' into 'useful'. The goal is reliable output: clear specs in, verifiable artifacts out.

8. The operator model: roles, trust levels, approvals

Treat your AI like a junior operator that can execute quickly, but needs explicit constraints and review gates for risky actions.

Recommended default roles

- Supervisor: plans work, reviews output, decides escalation.
- Worker: executes well-scoped tasks and produces artifacts.
- Auditor: checks output against acceptance criteria (tests, lint, diff review).

Trust ladder (simple)

- Level 0: read-only research and drafting.
- Level 1: write only inside workspace; no external network.
- Level 2: limited network (whitelist) + PR-based changes.
- Level 3: broader permissions only for isolated machines.

9. Workspace layout and files

A stable workspace layout reduces context drift and improves repeatability across sessions and agents.

Minimum set

- IDENTITY.md - what the agent is, what it is not, tone, constraints.
- MEMORY.md - durable facts, preferences, and project invariants.
- TASKS.md - backlog with statuses and links to artifacts.
- RUNBOOK.md - how to start/stop, verify, recover, and update.

Copy/paste templates are in the Appendix.

10. Prompting for work: specs, checklists, acceptance criteria

Most failures are unclear tasks. Use a simple structure that forces clarity.

```
Task: <one sentence>
Context: <what exists, what matters>
Inputs: <files/links>
Constraints: <time, tools, style>
Deliverables: <exact outputs>
Acceptance criteria: <how you will verify>
Risks: <what to avoid>
Escalate if: <conditions>
```

For coding tasks: require tests, lint, and a minimal reproduction before changing logic.

11. Guardrails: tools, secrets, and data handling

- Never paste long-lived API keys into prompts. Use env vars or secret files outside the repo.
- Use a dedicated machine or container for any agent with write + network permissions.
- Prefer allowlists over blocklists (domains, directories, commands).
- Log tool calls and keep an audit trail for changes.

12. Monitoring and quality control loops

Use a simple loop: plan -> execute -> verify -> package -> log -> repeat.

- Verification: tests pass, output meets spec, and files are where the buyer expects.
- Packaging: bundle artifacts, include a README, and note versions.
- Logging: write a short entry in CHANGELOG.md or TASKS.md with what changed.

Part III - Scale

This section is for higher throughput: multiple agents, batching, and business workflows where consistency matters more than cleverness.

13. Multi-agent patterns

Worker/Supervisor (recommended)

- Supervisor writes the plan and splits work into small tasks.
- Workers execute tasks independently and output artifacts.
- Auditor verifies artifacts and flags failures with concrete diffs.

Batching

- Use a spreadsheet/JSONL input file and process rows in chunks.
- Write one output file per item to avoid merge conflicts.
- Resume by checkpointing completed item IDs.

14. Documentation and handoff

- RUNBOOK.md: start/stop, update, verification commands, recovery steps.
- ARCHITECTURE.md: what talks to what, where configs live, where secrets live.
- CHANGELOG.md: human-readable log of changes.

15. Appendix: templates and copy/paste blocks

IDENTITY.md template

```
# IDENTITY.md

## Mission
You are an operational assistant that produces verifiable deliverables.

## Boundaries
- Do not act outside the workspace directory.
- Do not access private data unless explicitly provided.
- Do not run destructive commands.

## Output rules
- Prefer checklists and numbered steps.
- Provide commands in code blocks.
- State assumptions and how to verify them.

## Escalation
Escalate when:
- Requirements are ambiguous
- Actions require elevated permissions
- Changes affect production systems
```

MEMORY.md template

```
# MEMORY.md

## Stable facts
- Primary OS: <...>
- Primary model/provider: <...>
- Workspace path: <...>

## Preferences
- Tone: concise, minimal fluff
- Outputs: copy/paste commands, verification steps

## Invariants
- Secrets are environment variables
- Changes require tests before merge
```

TASKS.md template

```
# TASKS.md

## Active
- [ ] <task> (owner: agentX) -> artifact: <path>

## Done
- [x] <task> (date) -> artifact: <path>
```

RUNBOOK.md template

OpenClaw Super Guide v1.0

```
# RUNBOOK.md

## Start
- openclaw gateway start
- openclaw gateway status

## Verify
- openclaw doctor
- run smoke test task

## Update
- openclaw update
- restart gateway

## Recover
- check logs
- validate config
- rollback to last known good
```

Sales-friendly quick checklist (buyer)

- Pick your install path (Docker is easiest to reset).
- Set provider keys as environment variables (never in the PDF screenshots).
- Start gateway; run doctor; run the smoke test.
- Create workspace files (IDENTITY/MEMORY/TASKS/RUNBOOK).
- Run your first real task with acceptance criteria.