Multi-agent

Multi-Agent Routing

Goal: multiple *isolated* agents (separate workspace + `agentDir` + sessions), plus multiple channel accounts (e.g. two WhatsApps) in one running Gateway. Inbound is routed to an agent via bindings.

What is “one agent”?

An **agent** is a fully scoped brain with its own:

- **Workspace** (files, AGENTS.md/SOUL.md/USER.md, local notes, persona rules).
- **State directory** (`agentDir`) for auth profiles, model registry, and per-agent config.
- **Session store** (chat history + routing state) under `~/.openclaw/agents/<agentId>/sessions` .

Auth profiles are **per-agent**. Each agent reads from its own:

```
~/.openclaw/agents/<agentId>/agent/auth-profiles.json
```

Main agent credentials are **not** shared automatically. Never reuse `agentDir` across agents (it causes auth/session collisions). If you want to share creds, copy `auth-profiles.json` into the other agent’s `agentDir` .

Skills are per-agent via each workspace's `skills/` folder, with shared skills available from `~/.openclaw/skills`. See **Skills: per-agent vs shared**.

The Gateway can host **one agent** (default) or **many agents** side-by-side.

Workspace note: each agent's workspace is the **default cwd**, not a hard sandbox. Relative paths resolve inside the workspace, but absolute paths can reach other host locations unless sandboxing is enabled. See **Sandboxing**.

Paths (quick map)

- Config: `~/.openclaw/openclaw.json` (or `OPENCLAW_CONFIG_PATH`)
- State dir: `~/.openclaw` (or `OPENCLAW_STATE_DIR`)
- Workspace: `~/.openclaw/workspace` (or `~/.openclaw/workspace-<agentId>`)
- Agent dir: `~/.openclaw/agents/<agentId>/agent` (or `agents.list[].agentDir`)
- Sessions: `~/.openclaw/agents/<agentId>/sessions`

Single-agent mode (default)

If you do nothing, OpenClaw runs a single agent:

- `agentId` defaults to `main`.
- Sessions are keyed as `agent:main:<mainKey>`.
- Workspace defaults to `~/.openclaw/workspace` (or `~/.openclaw/workspace-<profile>` when `OPENCLAW_PROFILE` is set).
- State defaults to `~/.openclaw/agents/main/agent`.

Agent helper

Use the agent wizard to add a new isolated agent:

```
openclaw agents add work
```

Then add `bindings` (or let the wizard do it) to route inbound messages.

Verify with:

```
openclaw agents list --bindings
```

Quick start

1 Create each agent workspace

Use the wizard or create workspaces manually:

```
openclaw agents add coding
openclaw agents add social
```

Each agent gets its own workspace with `SOUL.md`, `AGENTS.md`, and optional `USER.md`, plus a dedicated `agentDir` and session store under `~/.openclaw/agents/<agentId>`.

2 Create channel accounts

Create one account per agent on your preferred channels:

Discord: one bot per agent, enable Message Content Intent, copy each token.

Telegram: one bot per agent via BotFather, copy each token.

WhatsApp: link each phone number per account.

```
openclaw channels login --channel whatsapp --account work
```

See channel guides: , , .

3 **Add agents, accounts, and bindings**

Add agents under `agents.list`, channel accounts under `channels.<channel>.accounts`, and connect them with `bindings` (examples below).

4 **Restart and verify**

```
openclaw gateway restart
openclaw agents list --bindings
openclaw channels status --probe
```

Multiple agents = multiple people, multiple personalities

With **multiple agents**, each `agentId` becomes a **fully isolated persona**:

- **Different phone numbers/accounts** (per channel `accountId`).
- **Different personalities** (per-agent workspace files like `AGENTS.md` and `SOUL.md`).
- **Separate auth + sessions** (no cross-talk unless explicitly enabled).

This lets **multiple people** share one Gateway server while keeping their AI “brains” and data isolated.

One WhatsApp number, multiple people (DM split)

You can route **different WhatsApp DMs** to different agents while staying on **one WhatsApp account**. Match on sender E.164 (like `+15551234567`) with `peer.kind: "direct"`. Replies still come from the same WhatsApp number (no per-agent sender identity).

Important detail: direct chats collapse to the agent's **main session key**, so true isolation requires **one agent per person**.

Example:

```
{
  agents: {
    list: [
      { id: "alex", workspace: "~/.openclaw/workspace-alex" },
      { id: "mia", workspace: "~/.openclaw/workspace-mia" },
    ],
  },
  bindings: [
    {
      agentId: "alex",
      match: { channel: "whatsapp", peer: { kind: "direct", id: "+15551230001" } }
    },
    {
      agentId: "mia",
      match: { channel: "whatsapp", peer: { kind: "direct", id: "+15551230002" } }
    },
  ],
  channels: {
    whatsapp: {
      dmPolicy: "allowlist",
      allowFrom: ["+15551230001", "+15551230002"],
    },
  },
}
```

Notes:

DM access control is **global per WhatsApp account** (pairing/allowlist), not per agent.

For shared groups, bind the group to one agent or use **Broadcast groups**.

Routing rules (how messages pick an agent)

Bindings are **deterministic** and **most-specific wins**:

1. `peer` match (exact DM/group/channel id)
2. `parentPeer` match (thread inheritance)
3. `guildId + roles` (Discord role routing)
4. `guildId` (Discord)
5. `teamId` (Slack)
6. `accountId` match for a channel
7. channel-level match (`accountId: "*"`)
8. fallback to default agent (`agents.list[].default` , else first list entry, default: `main`)

If multiple bindings match in the same tier, the first one in config order wins. If a binding sets multiple match fields (for example `peer` + `guildId`), all specified fields are required (`AND` semantics).

Multiple accounts / phone numbers

Channels that support **multiple accounts** (e.g. WhatsApp) use `accountId` to identify each login. Each `accountId` can be routed to a different agent, so one server can host multiple phone numbers without mixing sessions.

Concepts

- `agentId` : one “brain” (workspace, per-agent auth, per-agent session store).
- `accountId` : one channel account instance (e.g. WhatsApp account `"personal"` vs `"biz"`).
- `binding` : routes inbound messages to an `agentId` by `(channel, accountId, peer)` and optionally guild/team ids.

Direct chats collapse to `agent:<agentId>:<mainKey>` (per-agent “main”; `session.mainKey`).

Platform examples

Discord bots per agent

Each Discord bot account maps to a unique `accountId` . Bind each account to an agent and keep allowlists per bot.

```
{
  agents: {
    list: [
      { id: "main", workspace: "~/.openclaw/workspace-main" },
      { id: "coding", workspace: "~/.openclaw/workspace-coding" },
    ],
  },
  bindings: [
    { agentId: "main", match: { channel: "discord", accountId: "default" } },
    { agentId: "coding", match: { channel: "discord", accountId: "coding" } },
  ],
  channels: {
    discord: {
      groupPolicy: "allowlist",
      accounts: {
        default: {
          token: "DISCORD_BOT_TOKEN_MAIN",
          guilds: {
            "123456789012345678": {
              channels: {
                "222222222222222222": { allow: true, requireMention: false },
              },
            },
          },
        },
        coding: {
          token: "DISCORD_BOT_TOKEN_CODING",
          guilds: {
            "123456789012345678": {
              channels: {
                "333333333333333333": { allow: true, requireMention: false },
              },
            },
          },
        },
      },
    },
  },
}
```

Notes:

- Invite each bot to the guild and enable Message Content Intent.
- Tokens live in `channels.discord.accounts.<id>.token` (default account can use `DISCORD_BOT_TOKEN`).

Telegram bots per agent

```
{
  agents: {
    list: [
      { id: "main", workspace: "~/.openclaw/workspace-main" },
      { id: "alerts", workspace: "~/.openclaw/workspace-alerts" },
    ],
  },
  bindings: [
    { agentId: "main", match: { channel: "telegram", accountId: "default" } },
    { agentId: "alerts", match: { channel: "telegram", accountId: "alerts" } },
  ],
  channels: {
    telegram: {
      accounts: {
        default: {
          botToken: "123456:ABC...",
          dmPolicy: "pairing",
        },
        alerts: {
          botToken: "987654:XYZ...",
          dmPolicy: "allowlist",
          allowFrom: ["tg:123456789"],
        },
      },
    },
  },
}
```

Notes:

- Create one bot per agent with BotFather and copy each token.
- Tokens live in `channels.telegram.accounts.<id>.botToken` (default account can use `TELEGRAM_BOT_TOKEN`).

WhatsApp numbers per agent

Link each account before starting the gateway:

```
openclaw channels login --channel whatsapp --account personal
openclaw channels login --channel whatsapp --account biz
```

`~/.openclaw/openclaw.json` (JSON5):

```
{
  agents: {
    list: [
      {
        id: "home",
        default: true,
        name: "Home",
        workspace: "~/.openclaw/workspace-home",
        agentDir: "~/.openclaw/agents/home/agent",
      },
      {
        id: "work",
        name: "Work",
        workspace: "~/.openclaw/workspace-work",
        agentDir: "~/.openclaw/agents/work/agent",
      },
    ],
  },

  // Deterministic routing: first match wins (most-specific first).
  bindings: [
    { agentId: "home", match: { channel: "whatsapp", accountId: "personal" } },
    { agentId: "work", match: { channel: "whatsapp", accountId: "biz" } },

    // Optional per-peer override (example: send a specific group to work agent).
    {
      agentId: "work",
      match: {
        channel: "whatsapp",
        accountId: "personal",
        peer: { kind: "group", id: "1203630...@g.us" },
      },
    },
  ],

  // Off by default: agent-to-agent messaging must be explicitly enabled + allowli
  tools: {
    agentToAgent: {
      enabled: false,
      allow: ["home", "work"],
```

```
    },
  },

  channels: {
    whatsapp: {
      accounts: {
        personal: {
          // Optional override. Default: ~/.openclaw/credentials/whatsapp/personal
          // authDir: "~/.openclaw/credentials/whatsapp/personal",
        },
        biz: {
          // Optional override. Default: ~/.openclaw/credentials/whatsapp/biz
          // authDir: "~/.openclaw/credentials/whatsapp/biz",
        },
      },
    },
  },
}
```

Example: WhatsApp daily chat + Telegram deep work

Split by channel: route WhatsApp to a fast everyday agent and Telegram to an Opus agent.

```
{
  agents: {
    list: [
      {
        id: "chat",
        name: "Everyday",
        workspace: "~/.openclaw/workspace-chat",
        model: "anthropic/claude-sonnet-4-5",
      },
      {
        id: "opus",
        name: "Deep Work",
        workspace: "~/.openclaw/workspace-opus",
        model: "anthropic/claude-opus-4-6",
      },
    ],
  },
  bindings: [
    { agentId: "chat", match: { channel: "whatsapp" } },
    { agentId: "opus", match: { channel: "telegram" } },
  ],
}
```

Notes:

- If you have multiple accounts for a channel, add `accountId` to the binding (for example `{ channel: "whatsapp", accountId: "personal" }`).
- To route a single DM/group to Opus while keeping the rest on chat, add a `match.peer` binding for that peer; peer matches always win over channel-wide rules.

Example: same channel, one peer to Opus

Keep WhatsApp on the fast agent, but route one DM to Opus:

```
{
  agents: {
    list: [
      {
        id: "chat",
        name: "Everyday",
        workspace: "~/.openclaw/workspace-chat",
        model: "anthropic/claude-sonnet-4-5",
      },
      {
        id: "opus",
        name: "Deep Work",
        workspace: "~/.openclaw/workspace-opus",
        model: "anthropic/claude-opus-4-6",
      },
    ],
  },
  bindings: [
    {
      agentId: "opus",
      match: { channel: "whatsapp", peer: { kind: "direct", id: "+15551234567" } },
    },
    { agentId: "chat", match: { channel: "whatsapp" } },
  ],
}
```

Peer bindings always win, so keep them above the channel-wide rule.

Family agent bound to a WhatsApp group

Bind a dedicated family agent to a single WhatsApp group, with mention gating and a tighter tool policy:

```
{
  agents: {
    list: [
      {
        id: "family",
        name: "Family",
        workspace: "~/.openclaw/workspace-family",
        identity: { name: "Family Bot" },
        groupChat: {
          mentionPatterns: ["@family", "@familybot", "@Family Bot"],
        },
        sandbox: {
          mode: "all",
          scope: "agent",
        },
        tools: {
          allow: [
            "exec",
            "read",
            "sessions_list",
            "sessions_history",
            "sessions_send",
            "sessions_spawn",
            "session_status",
          ],
          deny: ["write", "edit", "apply_patch", "browser", "canvas", "nodes", "cr
        },
      },
    ],
  },
  bindings: [
    {
      agentId: "family",
      match: {
        channel: "whatsapp",
        peer: { kind: "group", id: "120363999999999999@g.us" },
      },
    },
```

```
    ],
```

Notes:

- Tool allow/deny lists are **tools**, not skills. If a skill needs to run a binary, ensure `exec` is allowed and the binary exists in the sandbox.
- For stricter gating, set `agents.list[].groupChat.mentionPatterns` and keep group allowlists enabled for the channel.

Per-Agent Sandbox and Tool Configuration

Starting with v2026.1.6, each agent can have its own sandbox and tool restrictions:

```
{
  agents: {
    list: [
      {
        id: "personal",
        workspace: "~/.openclaw/workspace-personal",
        sandbox: {
          mode: "off",  // No sandbox for personal agent
        },
        // No tool restrictions - all tools available
      },
      {
        id: "family",
        workspace: "~/.openclaw/workspace-family",
        sandbox: {
          mode: "all",     // Always sandboxed
          scope: "agent",  // One container per agent
          docker: {
            // Optional one-time setup after container creation
            setupCommand: "apt-get update && apt-get install -y git curl",
          },
        },
        tools: {
          allow: ["read"],                    // Only read tool
          deny: ["exec", "write", "edit", "apply_patch"],    // Deny others
        },
      },
    ],
  },
}
```

Note: `setupCommand` lives under `sandbox.docker` and runs once on container creation. Per-agent `sandbox.docker.*` overrides are ignored when the resolved scope is `"shared"`.

Benefits:

- **Security isolation**: Restrict tools for untrusted agents

- **Resource control**: Sandbox specific agents while keeping others on host
- **Flexible policies**: Different permissions per agent

Note: `tools.elevated` is **global** and sender-based; it is not configurable per agent. If you need per-agent boundaries, use `agents.list[].tools` to deny `exec`. For group targeting, use `agents.list[].groupChat.mentionPatterns` so @mentions map cleanly to the intended agent.

See **Multi-Agent Sandbox & Tools** for detailed examples.

‹ Compaction Presence ›

Powered by mintlify