Agent coordination

Sub-Agents

Sub-agents are background agent runs spawned from an existing agent run. They run in their own session (`agent:<agentId>:subagent:<uuid>`) and, when finished, **announce** their result back to the requester chat channel.

Slash command

Use `/subagents` to inspect or control sub-agent runs for the **current session**:

```
/subagents list
/subagents kill <id|#|all>
/subagents log <id|#> [limit] [tools]
/subagents info <id|#>
/subagents send <id|#> <message>
/subagents steer <id|#> <message>
/subagents spawn <agentId> <task> [--model <model>] [--thinking <level>]
```

`/subagents info` shows run metadata (status, timestamps, session id, transcript path, cleanup).

Spawn behavior

`/subagents spawn` starts a background sub-agent as a user command, not an internal relay, and it sends one final completion update back to the requester chat when the run finishes.

›

- The spawn command is non-blocking; it returns a run id immediately.
- On completion, the sub-agent announces a summary/result message back to the requester chat channel.
- For manual spawns, delivery is resilient:
  - OpenClaw tries direct `agent` delivery first with a stable idempotency key.
  - If direct delivery fails, it falls back to queue routing.
  - If queue routing is still not available, the announce is retried with a short exponential backoff before final give-up.
- The completion message is a system message and includes:
  - `Result` (`assistant` reply text, or latest `toolResult` if the assistant reply is empty)
  - `Status` (`completed successfully` / `failed` / `timed out`)
  - compact runtime/token stats
- `--model` and `--thinking` override defaults for that specific run.
- Use `info` / `log` to inspect details and output after completion.

Primary goals:

- Parallelize “research / long task / slow tool” work without blocking the main run.
- Keep sub-agents isolated by default (session separation + optional sandboxing).
- Keep the tool surface hard to misuse: sub-agents do **not** get session tools by default.
- Support configurable nesting depth for orchestrator patterns.

Cost note: each sub-agent has its **own** context and token usage. For heavy or repetitive tasks, set a cheaper model for sub-agents and keep your main agent on a higher-quality model. You can configure this via `agents.defaults.subagents.model` or per-agent overrides.

Tool

Use `sessions_spawn`:

- Starts a sub-agent run (`deliver: false`, global lane: `subagent`)
- Then runs an announce step and posts the announce reply to the requester chat channel
- Default model: inherits the caller unless you set `agents.defaults.subagents.model` (or per-agent `agents.list[].subagents.model`); an explicit `sessions_spawn.model` still wins.
- Default thinking: inherits the caller unless you set `agents.defaults.subagents.thinking` (or per-agent `agents.list[].subagents.thinking`); an explicit `sessions_spawn.thinking` still wins.

Tool params:

- `task` (required)
- `label?` (optional)
- `agentId?` (optional; spawn under another agent id if allowed)
- `model?` (optional; overrides the sub-agent model; invalid values are skipped and the sub-agent runs on the default model with a warning in the tool result)
- `thinking?` (optional; overrides thinking level for the sub-agent run)
- `runTimeoutSeconds?` (default `0`; when set, the sub-agent run is aborted after N seconds)

`cleanup?` (`delete|keep`, default `keep`)

Allowlist:

- `agents.list[].subagents.allowAgents`: list of agent ids that can be targeted via `agentId` (`["*"]` to allow any). Default: only the requester agent.

Discovery:

- Use `agents_list` to see which agent ids are currently allowed for `sessions_spawn`.

Auto-archive:

- Sub-agent sessions are automatically archived after `agents.defaults.subagents.archiveAfterMinutes` (default: 60).
- Archive uses `sessions.delete` and renames the transcript to `*.deleted.<timestamp>` (same folder).
- `cleanup: "delete"` archives immediately after announce (still keeps the transcript via rename).
- Auto-archive is best-effort; pending timers are lost if the gateway restarts.
- `runTimeoutSeconds` does **not** auto-archive; it only stops the run. The session remains until auto-archive.
- Auto-archive applies equally to depth-1 and depth-2 sessions.

Nested Sub-Agents

By default, sub-agents cannot spawn their own sub-agents (`maxSpawnDepth: 1`). You can enable one level of nesting by setting `maxSpawnDepth: 2`, which allows the **orchestrator pattern**: main → orchestrator sub-agent → worker sub-sub-agents.

How to enable

```
{
  agents: {
    defaults: {
      subagents: {
        maxSpawnDepth: 2, // allow sub-agents to spawn children (default: 1)
        maxChildrenPerAgent: 5, // max active children per agent session (default
        maxConcurrent: 8, // global concurrency lane cap (default: 8)
      },
    },
  },
}
```

Depth levels

Depth	Session key shape	Role	Can spawn?
0	`agent:<id>:main`	Main agent	Always
1	`agent:<id>:subagent:<uuid>`	Sub-agent (orchestrator when depth 2 allowed)	Only if `maxSpawnDepth >= 2`
2	`agent:<id>:subagent:<uuid>:subagent:<uuid>`	Sub-sub-agent (leaf worker)	Never

Announce chain

Results flow back up the chain:

1. Depth-2 worker finishes → announces to its parent (depth-1 orchestrator)
2. Depth-1 orchestrator receives the announce, synthesizes results, finishes → announces to main

3. Main agent receives the announce and delivers to the user

Each level only sees announces from its direct children.

›

Tool policy by depth

Depth 1 (orchestrator, when `maxSpawnDepth >= 2`): Gets `sessions_spawn`, `subagents`, `sessions_list`, `sessions_history` so it can manage its children. Other session/system tools remain denied.

Depth 1 (leaf, when `maxSpawnDepth == 1`): No session tools (current default behavior).

Depth 2 (leaf worker): No session tools — `sessions_spawn` is always denied at depth 2. Cannot spawn further children.

Per-agent spawn limit

Each agent session (at any depth) can have at most `maxChildrenPerAgent` (default: 5) active children at a time. This prevents runaway fan-out from a single orchestrator.

Cascade stop

Stopping a depth-1 orchestrator automatically stops all its depth-2 children:

`/stop` in the main chat stops all depth-1 agents and cascades to their depth-2 children.

`/subagents kill <id>` stops a specific sub-agent and cascades to its children.

`/subagents kill all` stops all sub-agents for the requester and cascades.

Authentication

Sub-agent auth is resolved by **agent id**, not by session type:

- The sub-agent session key is `agent:<agentId>:subagent:<uuid>`.
- The auth store is loaded from that agent's `agentDir`.
- The main agent's auth profiles are merged in as a **fallback**; agent profiles override main profiles on conflicts.

Note: the merge is additive, so main profiles are always available as fallbacks. Fully isolated auth per agent is not supported yet.

Announce

Sub-agents report back via an announce step:

- The announce step runs inside the sub-agent session (not the requester session).
- If the sub-agent replies exactly `ANNOUNCE_SKIP`, nothing is posted.
- Otherwise the announce reply is posted to the requester chat channel via a follow-up `agent` call (`deliver=true`).
- Announce replies preserve thread/topic routing when available (Slack threads, Telegram topics, Matrix threads).
- Announce messages are normalized to a stable template:
  - `Status:` derived from the run outcome (`success`, `error`, `timeout`, or `unknown`).
  - `Result:` the summary content from the announce step (or `(not available)` if missing).
  - `Notes:` error details and other useful context.
- `Status` is not inferred from model output; it comes from runtime outcome signals.

Announce payloads include a stats line at the end (even when wrapped):

- Runtime (e.g., `runtime 5m12s`)
- Token usage (input/output/total)
- Estimated cost when model pricing is configured (`models.providers.*.models[].cost`)
- `sessionKey`, `sessionId`, and transcript path (so the main agent can fetch history via `sessions_history` or inspect the file on disk)

Tool Policy (sub-agent tools)

By default, sub-agents get **all tools except session tools** and system tools:

- `sessions_list`
- `sessions_history`
- `sessions_send`
- `sessions_spawn`

When `maxSpawnDepth >= 2`, depth-1 orchestrator sub-agents additionally receive `sessions_spawn`, `subagents`, `sessions_list`, and `sessions_history` so they can manage their children.

Override via config:

```
{
  agents: {
    defaults: {
      subagents: {
        maxConcurrent: 1,
      },
    },
  },
  tools: {
    subagents: {
      tools: {
        // deny wins
        deny: ["gateway", "cron"],
        // if allow is set, it becomes allow-only (deny still wins)
        // allow: ["read", "exec", "process"]
      },
    },
  },
}
```

Concurrency

Sub-agents use a dedicated in-process queue lane:

- Lane name: `subagent`
- Concurrency: `agents.defaults.subagents.maxConcurrent` (default `8`)

Stopping

Sending `/stop` in the requester chat aborts the requester session and stops any active sub-agent runs spawned from it, cascading to nested children.

`/subagents kill <id>` stops a specific sub-agent and cascades to its children.

Limitations

- Sub-agent announce is **best-effort**. If the gateway restarts, pending “announce back” work is lost.
- Sub-agents still share the same gateway process resources; treat `maxConcurrent` as a safety valve.
- `sessions_spawn` is always non-blocking: it returns `{ status: "accepted", runId, childSessionKey }` immediately.
- Sub-agent context only injects `AGENTS.md` + `TOOLS.md` (no `SOUL.md`, `IDENTITY.md`, `USER.md`, `HEARTBEAT.md`, or `BOOTSTRAP.md`).
- Maximum nesting depth is 5 (`maxSpawnDepth` range: 1–5). Depth 2 is recommended for most use cases.
- `maxChildrenPerAgent` caps active children per session (default: 5, range: 1–20).

‹ Agent Send

Multi-Agent Sandbox & Tools ›