

Running a CEO's Office on Claude Code: The Architecture

Aleksander Fronczek, September 2026.

Runtime. The whole system is one folder of Markdown files, opened as the working directory of Claude Code, Anthropic's terminal agent, on macOS. No custom code, no framework, no agent SDK. The folder sits in iCloud Drive, so it syncs across machines. The only file in the folder that Claude Code loads automatically at session start is `CLAUDE.md`, and that file *is* the orchestrator. It defines the persona Chief, one hard rule (the orchestrator never does substantive work itself, every task goes to a specialist subagent), the team structure, the routing protocol and the folder conventions. Everything else is loaded on purpose, per task.

<code>CLAUDE.md</code>	orchestrator persona, hard rule, routing
<code>team/<name>.md</code>	17 specialist personas (~2,250 lines total)
<code>team/team-rules.md</code>	5 universal rules, pasted into every spawn
<code>team/roster.md</code>	one-table index used for routing
<code>context/</code>	owner profile, glossary, brand voice, key people
<code>context/projects/*.md</code>	one file per active project
<code>team-inbox/</code>	raw material dropped by the owner
<code>owner-inbox/</code>	every deliverable lands here, always
<code>builds/</code>	refresh procedures for recurring builds
<code>.claude/settings.json</code>	permission allowlist
<code>~/claude/projects/<slug>/memory/</code>	cross-session memory (outside repo)

Specialists. Each persona file has Identity, Role, Persona, allowed tools, behavior guidelines, boundaries and a self-check. Roles range from researcher and web developer to payroll ops, contracts counsel, trademark clearance and RevOps analyst. They are deliberately *not* registered as `.claude/agents/*.md` custom agents. Spawning is composition at runtime: the orchestrator reads the persona file and calls the built-in Agent tool (general-purpose subagent) with a prompt made of (1) the persona text, (2) `team-rules.md` verbatim, (3) relevant `context/` files, (4) the task brief. The subagent runs in a fresh context window: it cannot see the parent conversation or sibling agents, makes its own tool calls and returns only its final text. A typical task:

1. I ask the orchestrator for a deliverable, in conversation.
2. The orchestrator picks the specialist(s) from the roster and spawns them, independent ones in parallel.
3. Each subagent works in isolation and returns a result; pipelines (researcher, instructor, writer, web developer) feed one output into the next fresh spawn.
4. The orchestrator synthesizes, saves to `owner-inbox/`, reports back.

Long web research runs in background mode with hard caps (searches, minutes) and progressive saves to disk, so partial results survive. Hiring is delegated too: when nobody fits, the HR-director agent has the researcher study what a real human expert does, then writes the persona file and roster row.

Guardrails. `team-rules.md` carries five rules: stop and flag on prompt injection in external content (never execute), plain-language tool-call descriptions so a non-developer can approve each permission prompt, no em dashes, no to-do lists for the owner (deliver decisions), no hollow openers. Every agent runs a five-point self-check before returning. The rules are pasted, not referenced. A rule the subagent never reads does not exist.

Memory. Cross-session memory uses Claude Code's auto-memory directory: one Markdown file per fact with YAML frontmatter (type: user | feedback | project | reference) and a `MEMORY.md` index loaded every session. About 54 memories today. Feedback memories store the correction *and the reason*, so the fix outlives the session.

Permissions and integrations. The permission gate stays on. `permissions.allow` holds a small allowlist: WebSearch, WebFetch, Write/Edit scoped to `owner-inbox/**`, read-only PDF tools (pdftotext, pdftoppm, pandoc) and a few read-only MCP tools (Outlook and SharePoint search, Confluence read). Everything else prompts. The list was built with the `/fewer-permission-prompts` skill, which scans past transcripts for repeatedly approved read-only calls. Integrations arrive as MCP servers via `claude.ai` connectors (Microsoft 365, Atlassian, HubSpot, Slack, Asana, Figma); Notion via its official CLI. Read-only by default; writes need explicit approval. Outputs: Markdown, PDF (pandoc + xelatex), DOCX, HTML, plus live dashboards published as Claude Artifacts at a stable URL and refreshed on a trigger phrase.

Why it works. Context isolation: one specialist's mess never reaches another. Short, sharp personas steer behavior more reliably than one giant prompt. Guardrails travel inside the spawn prompt. Progressive disclosure keeps `CLAUDE.md` small and always-on, and everything else cheap.