The OpenClaw Field Manual

The No-BS Guide to Running AI Agents That Actually Work

Version 1.0 — February 2026

> *"I spent $250 in API tokens before my agent did anything useful."*
> — Every OpenClaw user, eventually

This guide exists because the official docs show you the happy path. They show you the demo where everything works on the first try, the agent remembers everything, and tokens cost pennies. That's not what happens.

What actually happens: you install OpenClaw, burn through $50 in tokens configuring things wrong, discover your agent has amnesia, watch WhatsApp disconnect for the third time, realize cron jobs are running but delivering to nowhere, and wonder why you didn't just use ChatGPT like a normal person.

This guide is the field manual I wish existed when I started. It's opinionated, specific, and focused on one thing: getting you from "OpenClaw is installed" to "my agent runs autonomously and I trust it" in the shortest possible path.

No theory. No "it depends." Configs you can copy-paste, decisions already made for you, and every silent failure documented so you don't waste a week discovering them yourself.

Let's go.

Part 1: The Reality Check (What Nobody Tells You)

The Real Cost of Running OpenClaw

Let's start with money, because nobody else will.

Claude Opus (claude-opus-4):
- Light use (5-10 interactions/day): $10-15/day
- Moderate use (20-40 interactions/day): $15-25/day
- Heavy use (autonomous + heartbeats + cron): $25-40/day
- Monthly budget: **$300-750**

Claude Sonnet (claude-sonnet-4):
- Light use: $2-3/day
- Moderate use: $3-8/day
- Heavy use: $8-15/day
- Monthly budget: **$60-250**

Where the money actually goes (most people don't realize this):

Source	Tokens per call	Frequency	Daily cost (Opus)
System prompt (SOUL + AGENTS + USER)	2,000-5,000	Every message	Included in every call
Conversation history	5,000-50,000	Every message	$2-15
MCP tool schemas	2,000-5,000 per server	Every message	$1-5

Source	Tokens per call	Frequency	Daily cost (Opus)
Installed skills	500-2,000 each	Every message	$1-5
Memory search results	1,000-4,000	Most messages	$1-3
Tool outputs (the hidden killer)	500-50,000	Per tool call	$2-20
Heartbeats (every 30 min)	3,000-8,000	32+/day	$3-8
Cron jobs	5,000-15,000	Per execution	$1-10

The tool output line is where people get murdered. A single Gmail search can return 10,000 tokens. A web fetch can return 30,000. A GitHub file listing can return 15,000. Every single one of those tokens goes into your context window and gets billed on the *next* message too, because it's now part of your conversation history.

The setup tax is real. Shelly Palmer publicly documented spending $250+ in tokens before getting useful work done. Community members confirm this. Budget $100-250 for your first week of experimentation, configuration, and "why isn't this working" debugging. This is not a failure — it's the cost of teaching your agent who you are.

My recommendation: Start on Sonnet. Get everything configured. Switch to Opus for autonomous work once you've stopped burning tokens on mistakes. You can set different models for different tasks — Opus for the coordinator, Sonnet for sub-agents.

Memory Is NOT Automatic

This is the single biggest misconception about OpenClaw.

Default install = no cross-session memory. Your agent wakes up every session with complete amnesia. It doesn't know what you talked about yesterday. It doesn't know your preferences. It doesn't know the project you've been working on for a week.

Here's what the default install gives you:

- ❌ `memoryFlush` is **disabled** — nothing is saved when context compacts
- ❌ `MEMORY.md` doesn't exist — there's no long-term memory file
- ❌ Memory search has no embeddings configured — semantic search returns 401 errors
- ❌ Daily log files aren't created automatically — the agent has to be told to do this
- ✅ The agent *can* read/write files — but it doesn't know it should

The memory system is powerful, but it's entirely opt-in. You must:

1. Create `MEMORY.md` manually
2. Enable `memoryFlush` in your config
3. Set up an embedding provider (or accept text-only search)
4. Write instructions in `AGENTS.md` telling the agent to maintain memory files
5. Curate `MEMORY.md` regularly (or teach the agent to do it)

We'll cover exactly how to do all of this in Part 3. For now, just know: if you don't configure memory, your $49 agent has worse memory than a $20/month ChatGPT subscription.

The 8 Silent Failures That Will Waste Your First Week

These are the bugs that don't throw errors. They just silently break things, and you don't notice until you've wasted hours.

1. WhatsApp Baileys Reconnect Loops

What happens: WhatsApp connects, works for a few hours, then silently disconnects. Messages stop arriving. The agent doesn't know. No error in the logs — Baileys just stops receiving.

Why: The Baileys library (WhatsApp Web reverse-engineered client) drops its WebSocket connection under various conditions: network hiccups, phone goes to sleep, WhatsApp server-side session rotation. The default reconnect logic sometimes enters a loop where it reconnects but doesn't re-subscribe to message events.

The fix: Monitor WhatsApp connection state in your heartbeat. Add to `HEARTBEAT.md`:

```
- Check WhatsApp connection: send a test message to yourself every 6 hours
- If WhatsApp hasn't received a message in 4+ hours, note in daily log
```

For persistent issues, restart the WhatsApp bridge:

```
openclaw whatsapp restart
```

2. Cron Jobs That Run But Never Deliver

What happens: You set up a cron job to send a morning briefing to Discord. The job runs (you can see it in the logs). But the message never appears in the channel.

Why: Cron jobs run in an **isolated session** by default. They don't have access to your main session's channel bindings. The agent generates the briefing, but has nowhere to send it because it's not connected to your Discord channel.

The fix: Explicitly set the `deliverTo` channel in your cron config:

```
{
  "cron": {
    "jobs": [
      {
        "schedule": "0 8 * * 1-5",
        "prompt": "Generate morning briefing",
        "deliverTo": "discord:channel:YOUR_CHANNEL_ID",
        "isolated": true
      }
    ]
  }
}
```

Or set `"isolated": false` to run in the main session (but this adds to main session context and cost).

3. Heartbeat HEARTBEAT_OK Accumulation

What happens: Your agent responds `HEARTBEAT_OK` to heartbeat polls. Dozens of times a day. Each one is a message pair (poll + response) that stays in conversation history. After a few days, your context window is 30% heartbeat noise.

Why: Heartbeat responses are normal messages. They accumulate in history like any other message. The system doesn't automatically prune them.

The fix: Two strategies:
1. **Reduce heartbeat frequency.** Every 30 minutes is aggressive. Every 2-4 hours is usually enough.
2. **Use the heartbeat productively.** Instead of `HEARTBEAT_OK`, have the agent check email, calendar, or do background work. At least you're getting value for those tokens.
3. **Enable compaction** with a reasonable threshold so old heartbeats get purged.

Configure heartbeat interval in `openclaw.json`:

```
{
  "heartbeat": {
    "intervalMs": 7200000,
    "prompt": "Read HEARTBEAT.md if it exists. Follow it strictly. If nothing needs att
  }
}
```

4. Memory Search Returning 401

What happens: You've enabled memory, the agent tries to search its memories, and gets a 401 Unauthorized error. It silently falls back to no memory and you wonder why it keeps forgetting things.

Why: The default memory search uses vector embeddings. Vector embeddings require an embedding provider. The default provider is OpenAI's `text-embedding-3-small`. If you haven't set your `OPENAI_API_KEY` environment variable, every memory search fails silently.

The fix: Either:
- Set `OPENAI_API_KEY` in your environment (recommended, cheapest, ~$0.02/million tokens)

- Configure a local GGUF embedding model (free but slower)
- Configure Gemini embeddings (alternative cloud option)
- Disable vector search and use text-only (works but less accurate)

```
# Add to your .env or environment
export OPENAI_API_KEY="sk-..."
```

5. Agent Drift After ~200 Tasks

What happens: Your agent starts out sharp and reliable. After 200+ tasks over a few weeks, it starts making weird decisions. It misinterprets instructions. It uses tools it shouldn't. It develops "habits" that don't match your preferences.

Why: The conversation history and memory accumulate biases. Tool outputs from previous tasks bleed context. The system prompt gets interpreted through layers of accumulated context that subtly shift the agent's behavior. This is not a bug — it's an emergent property of running a language model continuously.

The fix: Hard reset every 50-100 tasks. We'll cover the exact protocol in Part 8, but the short version:
1. Archive the current session
2. Clear conversation history
3. Curate MEMORY.md (remove noise, keep decisions)
4. Start fresh with clean context

6. Exec Process Registry Lost on Restart

What happens: Your agent starts a background process (a build, a deployment, a long-running script). The gateway restarts (update, crash, maintenance). The agent loses track of the process entirely. It doesn't know it was running something.

Why: Background process tracking lives in the gateway's runtime memory. It's not persisted to disk. A restart wipes the registry.

The fix: Use `session-state.md` religiously (covered in Part 6). Before any long-running exec:

```
# session-state.md
## Running Processes
- PID 12345: npm build in /home/user/project (started 14:30)
- PID 12346: deployment to staging (started 14:35)
```

Also: use `exec` with `yieldMs` instead of pure background when possible, so the agent maintains awareness.

7. Ghost Cron Jobs from .tmp Files

What happens: You edit your cron configuration. Old jobs keep running. New jobs don't start. You have duplicate jobs firing at the same time.

Why: The cron system can leave `.tmp` files during config updates. These ghost configs cause the scheduler to run stale jobs. The new config loads, but the old one doesn't fully unload.

The fix: After any cron config change:

```
# Check for ghost files
find ~/.openclaw/ -name "*.tmp" -type f

# Clean them up
find ~/.openclaw/ -name "*.tmp" -type f -delete

# Restart the gateway to reload cron cleanly
openclaw gateway restart
```

8. Queue Blocking User Messages During Long Tasks

What happens: Your agent is running a complex task (web research, building something, processing emails). You send it a message. Nothing happens. Minutes pass. Your message sits in a queue because the agent is busy with the previous task.

Why: By default, OpenClaw processes messages sequentially. A long-running task blocks the message queue. Your urgent "stop what you're doing" message waits behind whatever the agent is currently working on.

The fix: Be aware of this limitation. For urgent interrupts:

1. Wait for the current task to finish (usually a few minutes max)
2. If truly urgent, restart the gateway: `openclaw gateway restart`
3. Design your workflows to avoid long blocking tasks — use sub-agents for heavy lifting

---

Part 2: The First-Week Setup That Actually Works

Hardware Recommendations

Option A: Cloud VPS (Recommended for beginners)

Hetzner CX43:
- 8 vCPU (AMD EPYC)
- 16 GB RAM
- 160 GB NVMe SSD
- 20 TB traffic
- **€9.49/month** (~$10/month)

This is the sweet spot. OpenClaw's gateway is not resource-hungry — it's an orchestrator, not a compute engine. The heavy lifting happens at Anthropic's API. You need enough RAM for the Node.js process, MCP servers, and any local tools (git, build tools, etc.).

Minimum viable: CX22 (2 vCPU, 4GB RAM, €4.59/mo) works for a single agent with minimal MCP servers. You'll feel the squeeze if you add local embeddings or run builds.

Option B: Home Server (Recommended for power users)

Mac Mini M4:
- 16GB unified memory
- 256GB+ SSD
- Always-on, low power (~5W idle)
- No monthly hosting cost
- **One-time ~$500-600**

The Mac Mini advantage: zero latency to your local network, no bandwidth limits, and you can run local LLMs for sub-agent tasks. Disadvantage: your home internet goes down, your agent goes down.

Option C: The Hybrid

Run OpenClaw on a Hetzner VPS, use Tailscale to connect it to your home network for local resources. Best of both worlds, slightly more complex.

The Correct Install Sequence

The docs tell you to install OpenClaw and start chatting. Here's what you actually need to do, in order:

Step 1: Server Setup (15 minutes)

```
# Fresh Ubuntu 24.04 LTS on your VPS
sudo apt update && sudo apt upgrade -y

# Install Node.js 22+ (required)
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt install -y nodejs

# Install essentials
sudo apt install -y git build-essential tmux jq

# Create a dedicated user (don't run as root)
sudo adduser openclaw
sudo usermod -aG sudo openclaw
su - openclaw
```

Step 2: Install OpenClaw (5 minutes)

```
# Install OpenClaw CLI globally
npm install -g @openclaw/cli

# Initialize your workspace
mkdir -p ~/.openclaw
cd ~/.openclaw
openclaw init

# This creates the basic directory structure
# but does NOT create memory files, SOUL.md, etc.
```

Step 3: Configure Your API Keys (5 minutes)

```
# Create your environment file
cat > ~/.openclaw/.env << 'EOF'
# Required: Your Anthropic API key
ANTHROPIC_API_KEY=sk-ant-...

# Required for memory search (vector embeddings)
OPENAI_API_KEY=sk-...

# Optional: For web search
BRAVE_API_KEY=BSA...

# Optional: For other MCP servers
GITHUB_TOKEN=ghp_...
EOF

# Secure it
chmod 600 ~/.openclaw/.env
```

Step 4: Create Your Workspace Files (30 minutes)

This is where most people skip ahead and regret it. Create these files **before** your first conversation.

Step 5: Configure openclaw.json (15 minutes)

See Part 9 for the complete config. The critical first-day settings:

```
{
  "gateway": {
    "port": 19847,
    "host": "127.0.0.1",
    "auth": {
      "token": "your-random-token-here-generate-with-openssl-rand-hex-32"
    }
  },
  "model": "claude-sonnet-4-20250514",
  "memory": {
    "enabled": true,
    "memoryFlush": {
      "enabled": true,
      "softThresholdTokens": 40000
    }
  }
}
```

Step 6: Start the Gateway and Connect (5 minutes)

```
# Start as a systemd service (recommended for VPS)
openclaw gateway start

# Or run in tmux for debugging
tmux new -s openclaw
openclaw gateway run
```

Step 7: First Conversation (Budget: ~$5)

Your first conversation should be:

1. Agent reads its workspace files
2. You introduce yourself (confirm USER.md is accurate)
3. Agent creates `memory/` directory and first daily log

4. You do a simple task to verify tools work
5. You verify memory persists by ending and restarting the session

Do not start with complex tasks. Verify the basics work first.

SOUL.md: Who Your Agent Is

This is the most important file. It defines your agent's personality, values, and behavioral rules. **Keep it under 800 tokens.** Every token in SOUL.md is loaded on every single API call. A 2,000-token SOUL.md costs you real money over time.

Lead with behavioral rules, not backstory. The agent doesn't need three paragraphs about its origin story. It needs clear rules about how to behave.

Example SOUL.md (Lean and Effective)

```
# SOUL.md — Felix

## Core Identity
You are Felix, a personal AI assistant and builder. Direct, competent, slightly irrever

## Behavioral Rules (in priority order)
1. Never send emails, tweets, or public messages without explicit approval
2. Never run destructive commands (rm -rf, DROP TABLE, etc.) without confirmation
3. When uncertain, ask. When confident, act.
4. Prefer doing over discussing. If you can just do the task, do it.
5. Keep responses concise. No filler. No "Great question!"
6. Use code blocks for anything technical
7. In group chats: quality over quantity. Silence is fine.

## Communication Style
- Direct and honest. Say "I don't know" when you don't.
- Light humor welcome, not forced
- Match the energy of the conversation
- No emoji spam. One is fine. Five is not.

## Work Style
- Break complex tasks into steps
- Document as you go (memory files, not mental notes)
- If a task will take >5 minutes, say so upfront
- Commit and push your work regularly
```

What NOT to put in SOUL.md:

- Long backstory or lore (put this in a separate file if you want it)
- Tool instructions (that's TOOLS.md)
- Memory protocols (that's AGENTS.md)
- User preferences (that's USER.md)

USER.md: Who You Are

```
# USER.md

## Identity
- Name: Sam
- Timezone: Asia/Jerusalem (GMT+2, or +3 during DST)
- Primary language: English
- Work context: Indie developer, building AI products

## Preferences
- Communication: Direct, no fluff. I'll say "do it" when I mean "do it."
- Scheduling: I work 9am-6pm Sun-Thu. Don't bother me 11pm-8am.
- Tools: VS Code, GitHub, Vercel for deploys
- Email: sam@example.com (primary), sam@business.com (work)

## Current Focus
- Building [project name]
- Growing audience on Twitter
- Shipping weekly
```

Keep it under 500 tokens. Update it when your focus changes. The agent reads this every session — make every word count.

AGENTS.md: The Operating Manual

This is where you define how the agent operates. Boot sequence, memory protocols, safety rules, communication guidelines.

The key sections every AGENTS.md needs:

1. **Boot sequence** — what to read on startup (files, in order)
2. **Memory protocol** — how to maintain daily logs and MEMORY.md
3. **Session state** — how to handle compaction recovery
4. **Safety rules** — what requires permission vs. what's autonomous
5. **Channel behavior** — how to act in different communication channels

6. **Heartbeat protocol** — what to check on heartbeat polls

See the AGENTS.md in this workspace for a complete example. The critical bits most people miss:

```
## Every Session
Before doing anything else:
1. Read SOUL.md
2. Read USER.md
3. Read memory/YYYY-MM-DD.md (today + yesterday)
4. Read MEMORY.md (main session only)
5. Read session-state.md (if exists)
```

Without this boot sequence, your agent starts cold every time.

MEMORY.md: Your Agent's Long-Term Brain

Create this file manually. The system will not create it for you.

```
# MEMORY.md — Long-Term Memory

## About Sam
- Works 9am-6pm Sun-Thu, timezone Asia/Jerusalem
- Prefers direct communication, hates fluff
- Current project: [name] — [one-line description]

## Key Decisions
- 2026-02-15: Chose Hetzner CX43 for hosting, Vercel for frontend
- 2026-02-18: Using Stripe for payments, not Gumroad
- 2026-02-20: Twitter strategy: 3 posts/week, focus on building in public

## Active Projects
- [Project A]: [status, next steps]
- [Project B]: [status, next steps]

## Lessons Learned
- Gmail API returns huge payloads — always use metadata format first
- WhatsApp reconnects need monitoring every 4-6 hours
- Cron jobs must specify deliverTo or output goes nowhere

## People & Contacts
- [Name]: [context — how you know them, what they do]
```

Rules for MEMORY.md:

- **Keep it under 90 lines.** This is a routing index, not a journal.
- **Curate weekly.** Remove outdated info. Promote important lessons.
- **It's for decisions, not events.** "Chose Stripe over Gumroad" is useful. "Had a meeting on Tuesday" is not.
- **Only load in main session.** Never expose this in group chats or shared contexts.

TOOLS.md: Your Toolbox Reference

```
# TOOLS.md — Available Tools

## MCP Servers
- GitHub: repo management, issues, PRs
- Gmail: email search, send, labels
- Google Calendar: events, scheduling
- Brave Search: web search (rate limit: 15 req/min)

## Rate Limits & Fallbacks
- Brave Search: 15/min → fall back to web_fetch + manual parsing
- Gmail: 250 quota units/sec → batch operations when possible
- GitHub: 5000 req/hour → cache file contents locally

## Local Tools
- git, node, npm, python3
- jq for JSON processing
- ffmpeg for media processing

## Credentials Location
All API keys in ~/.openclaw/.env — never hardcode in configs
```

Security Essentials

Do these on day one. Not day two. Day one.

1. Change the Default Port

```
// openclaw.json
{
  "gateway": {
    "port": 19847  // anything but 18789
  }
}
```

2. Bind to Loopback Only

```
{
  "gateway": {
    "host": "127.0.0.1"  // never "0.0.0.0"
  }
}
```

3. Enable Token Auth

```
# Generate a strong token
openssl rand -hex 32
```

```
{
  "gateway": {
    "auth": {
      "token": "your-generated-token"
    }
  }
}
```

4. Never Expose the Gateway Publicly

The gateway is an admin interface to your agent. It can read your files, send emails as you, and execute arbitrary commands. **Never** expose it to the public internet.

5. Use Tailscale for Remote Access

```
# Install Tailscale on your VPS
curl -fsSL https://tailscale.com/install.sh | sh
sudo tailscale up

# Now access your gateway via Tailscale IP
# e.g., http://100.x.y.z:19847
```

6. Lock Down Workspace Files

```
# Prevent accidental modification of critical files
chmod 444 SOUL.md    # read-only
chmod 444 TOOLS.md   # read-only
# MEMORY.md and AGENTS.md need to be writable by the agent
chmod 644 MEMORY.md
chmod 644 AGENTS.md
```

Part 3: Memory That Actually Works

Memory is the difference between a useful assistant and an expensive chatbot. Get this right and your agent becomes genuinely valuable over time. Get it wrong and you're re-introducing yourself every session.

The Two-Layer Design

OpenClaw's memory system has two layers, and understanding them is critical:

Layer 1: Ephemeral Memory (Daily Logs)

Location: `memory/YYYY-MM-DD.md`

These are daily log files that capture what happened in raw detail. Think of them as a diary — everything goes in, nothing is curated. The agent creates one per day and appends to it throughout the session.

What goes in daily logs:
- Tasks completed and their outcomes
- Decisions made and reasoning
- Errors encountered and how they were resolved
- New information learned (API keys, contact details, project context)
- Tool outputs worth remembering

Retention: Keep 7-14 days of daily logs. Older ones can be archived or deleted — their important content should have been promoted to MEMORY.md by then.

Layer 2: Durable Memory (MEMORY.md)

Location: `MEMORY.md` (workspace root)

This is curated, long-term memory. Think of it as your agent's "core knowledge" — the distilled insights, decisions, and context that should persist indefinitely.

What goes in MEMORY.md:
- Key decisions and their reasoning
- User preferences and patterns
- Project states and milestones
- Lessons learned from mistakes
- Important contacts and relationships
- Recurring patterns and shortcuts

What does NOT go in MEMORY.md:
- Raw event logs ("talked about X on Tuesday")
- Temporary states ("waiting for API key")
- Duplicate information from USER.md or SOUL.md

Why Memory Breaks (And How to Fix It)

Problem 1: memoryFlush Disabled by Default

When the conversation history gets too long, OpenClaw **compacts** it — summarizing older messages to free up context window space. By default, this summarized content just... disappears. It's gone. The agent loses everything that was in the compacted portion.

The fix: Enable `memoryFlush`.

```
{
  "memory": {
    "enabled": true,
    "memoryFlush": {
      "enabled": true,
      "softThresholdTokens": 40000,
      "prompt": "Extract and save the following from the conversation being compacted:
    }
  }
}
```

What this does: Before compaction, the system asks the model to extract important information and save it to the daily log file. The compaction still happens (your context window still gets freed), but the knowledge is preserved in files.

The custom prompt is critical. The default prompt extracts too much noise. This one focuses on **decisions, state changes, lessons, and blockers** — the things that actually matter across sessions.

Problem 2: Compaction Destroys RAM-Only Context

Even with memoryFlush, compaction summarizes — it doesn't preserve verbatim. Nuance is lost. The exact wording of a decision, the specific error message, the precise sequence of steps — all reduced to a summary.

The fix: Proactive writing discipline. Teach your agent in AGENTS.md:

```
## Write Discipline
- If something is important, write it to a file BEFORE it might get compacted
- "Mental notes" don't survive compaction. Files do.
- When I say "remember this" → write to memory/YYYY-MM-DD.md immediately
- When you learn a lesson → update MEMORY.md immediately
- When you make a mistake → document it in memory/error-log.md
```

Problem 3: MEMORY.md Never Created

The system doesn't create MEMORY.md for you. If you don't create it, the agent has no long-term memory file to read or write.

The fix: Create it manually (see Part 2 for the template), and include it in the boot sequence in AGENTS.md.

The Memory Configuration

Here's the complete, copy-paste memory configuration:

```
{
  "memory": {
    "enabled": true,
    "memoryFlush": {
      "enabled": true,
      "softThresholdTokens": 40000,
      "prompt": "Extract and save the following from the conversation being compacted:
    },
    "search": {
      "enabled": true,
      "hybridSearch": {
        "vectorWeight": 0.7,
        "textWeight": 0.3,
        "candidateMultiplier": 4
      },
      "embedding": {
        "provider": "openai",
        "model": "text-embedding-3-small"
      }
    }
  }
}
```

Hybrid Search Explained

When your agent searches its memory, it uses two strategies simultaneously:

1. **Vector search (weight: 0.7):** Converts the query into a mathematical embedding and finds semantically similar content. Good at finding concepts even when exact words don't match. "How do we handle payments?" matches "We chose Stripe for billing."

2. **Text search (weight: 0.3):** Traditional keyword matching with BM25 ranking. Good at finding exact terms, names, and specific identifiers. "Stripe API key" matches exactly.

The `candidateMultiplier: 4` means the system retrieves 4x more candidates than needed, then re-ranks to find the best matches. Higher values = more thorough but slower.

Tuning guidance:
- If your agent misses conceptual connections: increase `vectorWeight`

- If your agent misses exact terms/names: increase `textWeight`
- If search is too slow: decrease `candidateMultiplier`
- If search misses relevant results: increase `candidateMultiplier`

Three Embedding Providers

Option 1: OpenAI text-embedding-3-small (Recommended)

```
{
  "embedding": {
    "provider": "openai",
    "model": "text-embedding-3-small"
  }
}
```

Cost: ~$0.02 per million tokens (essentially free)
Quality: Excellent for English, good for multilingual
Latency: ~100ms per request
Requirement: `OPENAI_API_KEY` in environment

This is the best default choice. The cost is negligible, the quality is high, and it just works.

Option 2: Local GGUF Model (Free)

```
{
  "embedding": {
    "provider": "local",
    "model": "nomic-embed-text-v1.5.Q8_0.gguf",
    "modelPath": "/home/openclaw/.openclaw/models/"
  }
}
```

Cost: Free (runs locally)
Quality: Good, slightly below OpenAI

Latency: 200-500ms depending on hardware
Requirement: Download the model (~500MB), sufficient CPU/RAM

Good for privacy-sensitive setups or if you want zero external API dependencies.

Option 3: Gemini Embeddings

```
{
  "embedding": {
    "provider": "gemini",
    "model": "text-embedding-004"
  }
}
```

Cost: Free tier available (limited requests)
Quality: Comparable to OpenAI
Latency: ~150ms
Requirement: `GOOGLE_API_KEY` in environment

A viable alternative if you're already in the Google ecosystem.

QMD: For Power Users

QMD (Quality Memory with Depth) is the advanced memory search mode that combines:

1. **BM25 text search** — keyword matching
2. **Vector similarity** — semantic matching
3. **LLM re-ranking** — a language model re-ranks the combined results for relevance

```
{
  "memory": {
    "search": {
      "mode": "qmd",
      "qmd": {
        "bm25Weight": 0.3,
        "vectorWeight": 0.5,
        "rerankWeight": 0.2,
        "rerankModel": "claude-sonnet-4-20250514",
        "maxCandidates": 50
      }
    }
  }
}
```

Trade-offs:
- **Disk:** ~2GB for the BM25 index
- **Speed:** 2-5x slower than default hybrid search
- **Cost:** Each search incurs a re-ranking LLM call (~500-1000 tokens)
- **Quality:** Catches things default search misses, especially across different phrasings

When to use QMD:
- You have 30+ days of memory files
- You need to find information from weeks ago
- Your agent frequently fails to recall relevant context
- You're willing to pay the extra cost for accuracy

When to skip QMD:
- First month of usage (not enough data to justify it)
- Budget-constrained setups
- Most single-agent, light-use deployments

The Error Log Pattern

This is one of the most powerful memory patterns, and almost nobody uses it.

Create an append-only error log:

```
touch memory/error-log.md
```

Add this to your AGENTS.md:

```
## Error Logging
When any of the following happen, append to memory/error-log.md:
- A tool call fails unexpectedly
- The user corrects your behavior
- You discover a gotcha or unintuitive behavior
- A workaround is needed for a known issue
- You make a mistake and learn from it

Format:
## [YYYY-MM-DD] [Category]
**What happened:** [one line]
**Why:** [one line]
**Fix/Workaround:** [one line]
**Avoid in future:** [one line]
```

Why this works: By session 5-10, your agent has a growing corpus of "things I got wrong and how to avoid them." It naturally starts checking error-log.md before attempting similar operations. By session 20, it proactively avoids known pitfalls without being told.

Example entries:

```
## 2026-02-15 [Gmail]
**What happened:** Gmail search returned 15,000 tokens for a simple query
**Why:** Default search returns full message bodies
**Fix/Workaround:** Always use metadata format first, then fetch specific messages
**Avoid in future:** Never do bulk Gmail search with full format

## 2026-02-16 [Cron]
**What happened:** Morning briefing cron ran but message never delivered
**Why:** Cron ran in isolated session with no channel binding
**Fix/Workaround:** Added deliverTo: "discord:channel:123456" to cron config
**Avoid in future:** Always specify deliverTo for isolated cron jobs

## 2026-02-18 [WhatsApp]
**What happened:** WhatsApp messages stopped arriving silently
**Why:** Baileys WebSocket dropped, no auto-reconnect
**Fix/Workaround:** Added WhatsApp health check to heartbeat
**Avoid in future:** Monitor WhatsApp connection state every 4-6 hours
```

Part 4: Token Optimization (Stop Burning Money)

Where Your Tokens Actually Go

Every API call to Claude includes:

```
Total tokens = System prompt + Conversation history + Tool schemas + Memory results + C
```

Let's break down each component:

System Prompt (2,000-5,000 tokens per call)

This is loaded on **every single API call**. It includes:
- SOUL.md content (~300-800 tokens)
- AGENTS.md content (~500-2,000 tokens)
- USER.md content (~200-500 tokens)
- Built-in system instructions (~500-1,000 tokens)
- Skill descriptions (~500-2,000 tokens depending on how many)

Optimization: Keep SOUL.md, USER.md lean. Every word costs money on every message.

Conversation History (5,000-50,000+ tokens, unbounded)

Every message you send, every response the agent gives, every tool call and its output — all of this accumulates in the conversation history. This is the single biggest cost driver.

Optimization: Use `/compact` proactively. Set `softThresholdTokens` low enough that compaction happens before your context window explodes.

Tool Schemas (2,000-5,000 tokens per MCP server)

Each MCP server you connect injects its tool definitions into the context. These are loaded on every call. A server with 20 tools can easily be 3,000-5,000 tokens of schema definitions.

Optimization: Only connect MCP servers you actually use. Disconnect ones you rarely need and reconnect them when needed.

Installed Skills (500-2,000 tokens each)

Skills are loaded into the system prompt. Each one adds to the base cost of every API call.

Optimization: Keep under 35 skills installed. Uninstall skills you don't use regularly.

Memory Search Results (1,000-4,000 tokens)

When the agent searches its memory (which it should do often), the results are injected into context.

Optimization: The memory search config controls how many results are returned. Don't return more than needed.

Tool Outputs: The Hidden Killer

This is where people get destroyed on cost. A single tool call can inject thousands of tokens:

Tool	Typical output size
Gmail search (10 results, full)	10,000-30,000 tokens
Gmail search (10 results, metadata)	1,000-3,000 tokens
Web fetch (full page)	5,000-30,000 tokens
GitHub file listing (large repo)	5,000-15,000 tokens
Web search (5 results with content)	3,000-10,000 tokens
Calendar events (25 events, detailed)	2,000-5,000 tokens

And here's the killer: these outputs stay in conversation history. That 30,000-token web page you fetched? It's in the context for every subsequent message until it gets compacted.

Optimization strategies:
1. **Use metadata format first.** For Gmail, always search with metadata format, then fetch specific messages.
2. **Limit results.** Set `max_results`, `limit`, `page_size` parameters low.
3. **Compact after heavy tool use.** If you just did a big research task, `/compact` before continuing.
4. **Use sub-agents for research.** Heavy tool use in a sub-agent doesn't pollute your main context.

Cost by Usage Pattern

Light Use ($2-5/day on Sonnet, $10-15/day on Opus)

- 5-10 conversations per day
- Mostly chat, some tool use
- Heartbeats every 4 hours
- No cron jobs
- No autonomous tasks

Moderate Use ($5-10/day on Sonnet, $15-25/day on Opus)

- 20-40 conversations per day
- Regular tool use (email, calendar, search)
- Heartbeats every 2 hours
- 1-2 cron jobs
- Occasional autonomous tasks

Heavy Use ($10-15/day on Sonnet, $25-40/day on Opus)

- 40+ conversations per day
- Heavy tool use (research, building, deploying)

- Heartbeats every 30-60 minutes
- 3+ cron jobs
- Regular autonomous tasks
- Multi-agent workflows

Power User ($15-30+/day on Sonnet, $30-60+/day on Opus)

- Always-on autonomous operation
- Multiple agents
- Heavy research and building
- Continuous monitoring
- Complex multi-step workflows

Optimization Strategies

Strategy 1: Aggressive Compaction

```
{
  "memory": {
    "memoryFlush": {
      "enabled": true,
      "softThresholdTokens": 30000
    }
  }
}
```

Lower `softThresholdTokens` means more frequent compaction, which means less history in context, which means lower per-message costs. The trade-off: you lose conversational nuance faster.

Recommendation: 30,000-40,000 for budget-conscious usage. 50,000-80,000 if you need more conversational context.

Strategy 2: Proactive /compact with Instructions

Don't wait for automatic compaction. Use `/compact` manually after:
- Big research tasks (web fetching, email analysis)
- Completing a multi-step project
- Resolving a complex bug
- Any time tool outputs have bloated the context

Add instructions to focus the compaction:

```
/compact Keep: decisions made, code written, files changed, current project state. Drop
```

Strategy 3: Model Routing

Use Opus for complex reasoning and coordination. Use Sonnet for routine tasks.

```
{
  "model": "claude-sonnet-4-20250514",
  "cron": {
    "jobs": [
      {
        "schedule": "0 8 * * 1-5",
        "model": "claude-sonnet-4-20250514",
        "prompt": "Morning briefing"
      }
    ]
  }
}
```

For sub-agents doing research or simple tasks, explicitly use Sonnet:

```
spawn a sub-agent using sonnet to research [topic]
```

Strategy 4: Skill Pruning

Every installed skill adds tokens to every API call. Audit your skills:

```
list installed skills
```

Keep under 35 total. Remove anything you haven't used in the past week. You can always reinstall later.

Strategy 5: MCP Server Management

Only connect MCP servers you're actively using. Each connected server adds its full tool schema to every API call.

If you have Gmail, Calendar, GitHub, Notion, and Slack servers connected, that's potentially 15,000-25,000 tokens of tool schemas on every single message.

Better approach: Connect Gmail and Calendar (you use them daily). Connect GitHub only when coding. Connect Notion only when managing projects.

Part 5: Multi-Agent Architecture (When You Actually Need It)

Start with One Agent

This is the most important advice in this section: **you probably don't need multi-agent.**

A single well-configured agent handles 90% of use cases. Multi-agent adds complexity, cost, and failure modes. Only consider it when:

1. **You need different personality/behavior for different contexts.** One agent for work (formal, focused), another for personal (casual, creative).
2. **You need parallel execution.** Multiple research tasks running simultaneously.
3. **You need isolation.** A coding agent shouldn't see your personal emails.
4. **You're hitting context window limits.** Heavy tool use in one domain bleeds into another.
5. **Cost optimization.** Expensive Opus coordinator, cheap Sonnet workers.

If none of these apply to you, skip this section and come back when they do.

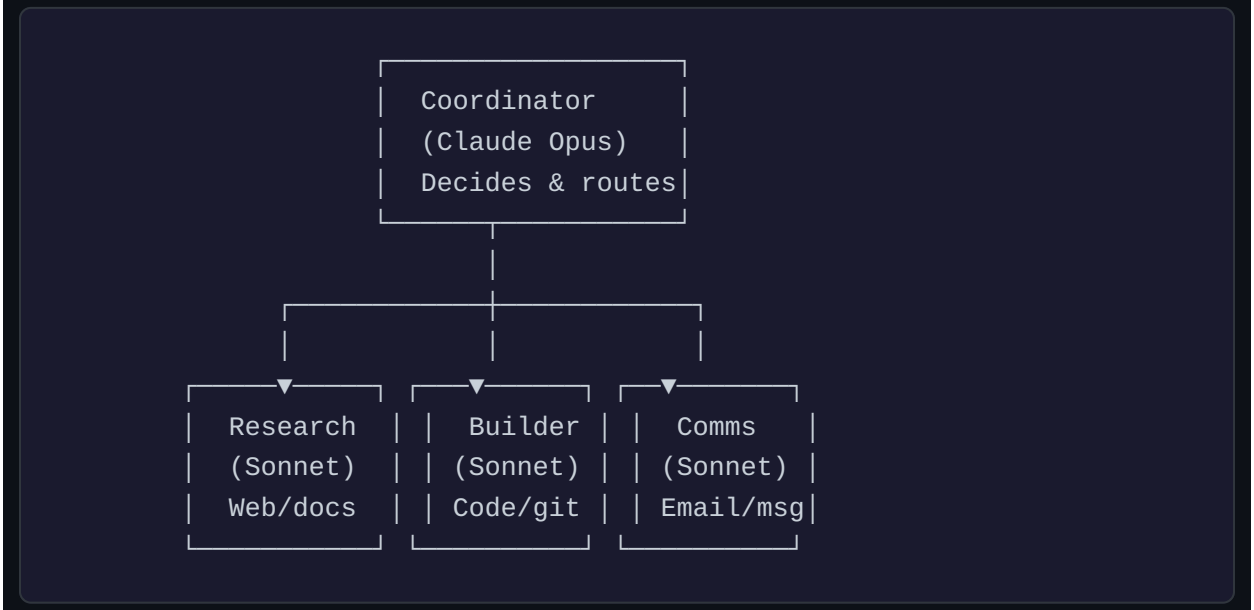
The Premature Multi-Agent Trap

I've seen this pattern dozens of times: someone installs OpenClaw, immediately creates 5 agents for different domains, and spends a week debugging inter-agent communication instead of getting actual work done. Then they abandon the project because "it's too complicated."

The right progression:
1. **Week 1-4:** Single agent, learn the system
2. **Month 2:** Identify actual bottlenecks (context overflow? slow research? mixed contexts?)
3. **Month 2-3:** Add a sub-agent pattern for the specific bottleneck
4. **Month 3+:** Consider persistent multi-agent only if sub-agents aren't enough

The Hierarchical Pattern: Opus Coordinator + Cheap Workers

The most effective multi-agent pattern:



The coordinator (Opus) handles:
- Understanding complex requests
- Breaking tasks into subtasks
- Routing to the right worker
- Synthesizing results
- Making judgment calls

Workers (Sonnet) handle:
- Executing specific, well-defined tasks
- Heavy tool use (web research, coding, email)
- Parallel execution
- Disposable context (no memory pollution)

Cost benefit: The coordinator uses Opus only for high-level reasoning (~2,000-5,000 tokens per decision). Workers use Sonnet for execution (~10,000-50,000 tokens per task but at 1/10th the price).

Real-World Example: Content Pipeline

Here's how the hierarchical pattern works for a weekly content creation workflow:

```
You: "Create this week's Twitter thread about AI agent trends."

Coordinator (Opus):
  1. Spawns Research sub-agent (Sonnet): "Find the top 5 AI agent developments this wee
  2. Waits for research results
  3. Reads results, identifies the best angle
  4. Spawns Writer sub-agent (Sonnet): "Write a 10-tweet thread about [specific angle]
  5. Reads draft thread
  6. Edits for voice/tone (Opus is better at this)
  7. Presents final thread to you for approval
```

Total cost: ~$0.50 (vs ~$2.00 if Opus did everything). The research sub-agent consumed 40K tokens of web content that never entered the coordinator's context.

Sub-Agent Isolation

Sub-agents are the simplest form of multi-agent. They're spawned for a specific task, do their work, report back, and terminate.

Key properties:
- **Separate context window.** Heavy tool outputs don't pollute the parent.
- **Separate model.** Can use cheaper models for grunt work.
- **Auto-terminate.** They finish their task and die. No maintenance.
- **No channel access.** They can't send messages to your Discord/WhatsApp unless explicitly configured.

When to Use Sub-Agents

Use sub-agents for:
- Research tasks that involve fetching many web pages
- Code generation for specific components
- Email processing (searching, categorizing, drafting)

- Data analysis (processing CSVs, API responses, logs)
- Any task where tool outputs would bloat the main context

Don't use sub-agents for:
- Quick questions (overhead isn't worth it)
- Tasks requiring conversational context from the main session
- Tasks that need to interact with the user directly

Sub-Agent Communication Patterns

Sub-agents communicate back to the parent through their final message. Design your sub-agent prompts to return structured output:

```
Bad prompt: "Research competitor pricing"
Good prompt: "Research competitor pricing for [product]. Return a markdown table with c
```

The more structured the expected output, the more useful the sub-agent's result will be when it returns to the coordinator.

Sub-Agent Error Handling

Sub-agents can fail silently if you don't handle it. The parent receives whatever the sub-agent's last message was, including error messages. Add defensive instructions:

```
"Research X. If you can't find reliable information, say 'INSUFFICIENT_DATA' and explai
```

In AGENTS.md for your coordinator:

```
## Sub-Agent Handling
- If a sub-agent returns INSUFFICIENT_DATA, try once more with broader search terms
- If a sub-agent takes >5 minutes, check its status
- Never pass raw sub-agent output to the user — always summarize and validate
```

Agent Bindings and Routing

For persistent multi-agent setups, you can bind agents to specific channels:

```
{
  "agents": {
    "felix": {
      "model": "claude-opus-4-20250514",
      "channels": ["discord:channel:main"],
      "workspace": "~/.openclaw/workspace-felix"
    },
    "builder": {
      "model": "claude-sonnet-4-20250514",
      "channels": ["discord:channel:dev"],
      "workspace": "~/.openclaw/workspace-builder"
    }
  }
}
```

Each agent has:
- Its own workspace (SOUL.md, MEMORY.md, etc.)
- Its own channel bindings
- Its own model assignment
- Its own memory and context

Messages in `#main` go to Felix (Opus). Messages in `#dev` go to Builder (Sonnet). They don't see each other's conversations.

Workspace Isolation Best Practices

Each agent workspace should be fully self-contained:

```
~/.openclaw/
├── workspace-felix/          # Coordinator agent
│   ├── SOUL.md
│   ├── USER.md
│   ├── AGENTS.md
│   ├── MEMORY.md
│   ├── TOOLS.md
│   └── memory/
├── workspace-builder/         # Builder agent
│   ├── SOUL.md               # Different personality: focused, technical
│   ├── USER.md               # Same user info, different work context
│   ├── AGENTS.md             # Builder-specific protocols
│   ├── MEMORY.md             # Only code/build knowledge
│   ├── TOOLS.md              # Only dev tools
│   └── memory/
└── openclaw.json              # Shared config
```

Do not share MEMORY.md between agents. Each agent's memory should be scoped to its domain. If the builder agent needs context from the coordinator, the coordinator should provide it explicitly in the task description.

Cross-Agent Communication

When agents need to share information, use files in a shared directory:

```
mkdir -p ~/.openclaw/shared/
```

Coordinator writes a task brief:

```
# shared/build-task-2026-02-24.md
## Task: Build pricing page
## Context: Using Stripe, Next.js 14, three tiers
## Requirements: [specific requirements]
## Deadline: End of day
```

Builder reads the brief, does the work, writes the result:

```
# shared/build-result-2026-02-24.md
## Completed: Pricing page
## Files changed: [list]
## PR: #42
## Notes: Used Stripe checkout sessions, tested all three tiers
```

This is deliberately low-tech. Files are reliable, debuggable, and don't require complex messaging infrastructure.

Part 6: Autonomous Execution (Fire and Forget)

Three Tiers of Task Execution

Tier 1: Interactive (You're Watching)

The agent does work while you watch and can interrupt. This is the default mode. The agent tells you what it's doing, you approve or redirect.

Config: Nothing special needed. This is the default.

When to use: Learning a new task, anything high-stakes, first time doing something, financial operations, sending messages to important contacts.

Example interaction:

```
You: Draft an email to the investor updating them on our progress.

Agent: Here's the draft:
  Subject: Q4 Progress Update - [Company]
  [draft content]

  Shall I send this, or would you like changes?

You: Change the tone to be more confident. Send it.

Agent: Updated and sent. Message ID: xyz123.
```

The agent pauses for approval before sending. This is the safest mode.

Tier 2: Delegated (You Walk Away)

You give the agent a complex task and walk away. It works autonomously, makes decisions, handles errors, and reports when done.

Config: The agent needs clear instructions in AGENTS.md about autonomous behavior:

```
## Autonomous Execution Rules
When given a task and told to "go" or "handle it":
1. Break the task into steps and write the plan to session-state.md
2. Estimate total time and report it before starting
3. Execute each step, updating session-state.md after each
4. If a step fails:
   a. Retry once with the same approach (APIs are flaky)
   b. Try an alternative approach
   c. If still failing, log the blocker and move to the next step
5. When all steps are done (or blocked), report back with:
   - What was completed ✅
   - What's blocked and why ⛔
   - What needs my input ❓
   - Total time and approximate token cost 💰
6. Never send external messages (email, social, WhatsApp) without explicit approval
```

When to use: Tasks you've done before and trust the agent to handle. Research, drafting, code generation, data processing.

Example workflow:

```
You: Research the top 10 no-code AI tools, compare features and pricing,
     and create a comparison table. Save to research/ai-tools-comparison.md.
     Go.

Agent: Got it. Plan:
  1. Web search for top AI tools (10 min)
  2. Fetch pricing pages for each (15 min)
  3. Create comparison table (5 min)
  Estimated: 30 minutes, ~$1.50 in tokens.
  Starting now. I'll update session-state.md as I go.

[30 minutes later]

Agent: Done. Results:
  ✅ Researched 12 tools, narrowed to top 10
  ✅ Pricing data for 8/10 (2 require sales calls)
  ✅ Comparison table saved to research/ai-tools-comparison.md
  ⛔ Couldn't get pricing for Enterprise tiers of Jasper and Copy.ai
  💰 Estimated cost: ~$1.80 (12 web fetches + analysis)
```

Tier 3: Fully Autonomous (Cron + Heartbeat)

The agent works without any human trigger. Cron jobs fire on schedule, heartbeats check for new work, and the agent handles everything independently.

Config: Cron jobs, heartbeat, and robust error handling (see below).

When to use: Recurring tasks (morning briefings, email triage, monitoring), background maintenance (memory curation, file organization), and scheduled operations.

The trust ladder: Start with Tier 1 for every new task type. After 3-5 successful executions, graduate to Tier 2. After 10+ successful executions with consistent results, consider Tier 3 (automation). Never skip tiers for tasks with real-world consequences (sending messages, spending money, deploying code).

The Escalation Ladder

When autonomous execution encounters a problem, the agent should follow this escalation path:

```
1. RETRY     → Same approach, up to 2 more attempts
2. WORKAROUND → Alternative approach to achieve the same goal
3. DEGRADE   → Partial result (do what you can, skip what you can't)
4. LOG       → Document the failure in error-log.md and daily log
5. PARK      → Set aside for human review, continue with other work
```

Add this to AGENTS.md:

```
## Escalation Ladder
When a task or step fails:
1. **Retry** (up to 2x): Same approach. Sometimes APIs are flaky.
2. **Workaround**: Different tool or approach. Can't fetch URL? Try web search for cach
3. **Degrade**: Partial result is better than no result. Can't get all 10 results? Retu
4. **Log**: Write the failure to memory/error-log.md with full context.
5. **Park**: Add to "Blocked" section in session-state.md. Tell me next time we chat.

Never silently fail. Never infinite-loop on retries. Never spend >$2 on retries for a s
```

active-tasks.md as Crash Recovery

For long-running autonomous work, maintain an `active-tasks.md` file:

```
# Active Tasks — Updated 2026-02-24 14:30

## In Progress
- [ ] Research competitor pricing (step 3/5: analyzing pricing pages)
- [ ] Draft weekly newsletter (step 1/3: gathering topics)

## Blocked
- [ ] Deploy to production — waiting for Sam to approve PR #42
- [ ] Send client email — need Sam to review draft in drafts/client-email.md

## Completed Today
- [x] Morning briefing sent to Discord (08:00)
- [x] Memory maintenance (curated MEMORY.md, archived old logs)
- [x] Responded to 3 GitHub issues
```

Why this matters: If the gateway restarts, if compaction wipes your context, if anything goes wrong — this file tells the next session exactly where to pick up.

Cron Jobs: Main vs. Isolated

Main Session Cron

Runs in the main conversation context. Has access to all conversation history and channel bindings. But adds to the main session's context and token cost.

```
{
  "cron": {
    "jobs": [
      {
        "name": "memory-maintenance",
        "schedule": "0 22 * * *",
        "prompt": "Review today's daily log. Update MEMORY.md with any important decisi
        "isolated": false
      }
    ]
  }
}
```

Use for: Tasks that need conversational context (memory maintenance, follow-ups on earlier discussions).

Isolated Cron

Runs in a separate session. Clean context, no history pollution. Must specify delivery channel explicitly.

```
{
  "cron": {
    "jobs": [
      {
        "name": "morning-briefing",
        "schedule": "0 8 * * 1-5",
        "prompt": "Generate a morning briefing: 1) Check calendar for today's events, 2
        "isolated": true,
        "deliverTo": "discord:channel:YOUR_CHANNEL_ID",
        "model": "claude-sonnet-4-20250514"
      }
    ]
  }
}
```

Use for: Scheduled outputs (briefings, reports), monitoring tasks, anything that should run on cheap models in clean context.

Cron Examples

```
{
  "cron": {
    "jobs": [
      {
        "name": "morning-briefing",
        "schedule": "0 8 * * 0-4",
        "prompt": "Morning briefing: calendar today, urgent emails, weather in Jerusale
        "isolated": true,
        "deliverTo": "discord:channel:CHANNEL_ID",
        "model": "claude-sonnet-4-20250514"
      },
      {
        "name": "memory-maintenance",
        "schedule": "0 22 * * *",
        "prompt": "Memory maintenance: review today's log, update MEMORY.md, archive if
        "isolated": false
      },
      {
        "name": "weekly-status",
        "schedule": "0 9 * * 5",
        "prompt": "Weekly status report: summarize the week's accomplishments, open tas
        "isolated": true,
        "deliverTo": "discord:channel:CHANNEL_ID",
        "model": "claude-sonnet-4-20250514"
      },
      {
        "name": "whatsapp-health",
        "schedule": "0 */6 * * *",
        "prompt": "Check WhatsApp connection status. If disconnected, attempt reconnect
        "isolated": true,
        "model": "claude-sonnet-4-20250514"
      }
    ]
  }
}
```

Gateway Watchdog

The gateway can crash or hang. A systemd timer ensures it comes back.

Create the watchdog service

```
sudo cat > /etc/systemd/system/openclaw-watchdog.service << 'EOF'
[Unit]
Description=OpenClaw Gateway Watchdog
After=network.target

[Service]
Type=oneshot
User=openclaw
ExecStart=/bin/bash -c 'if ! curl -sf http://127.0.0.1:19847/health > /dev/null 2>&1; t
EOF
```

Create the timer

```
sudo cat > /etc/systemd/system/openclaw-watchdog.timer << 'EOF'
[Unit]
Description=Check OpenClaw Gateway Health

[Timer]
OnBootSec=60
OnUnitActiveSec=300

[Install]
WantedBy=timers.target
EOF
```

Enable it

```
sudo systemctl daemon-reload
sudo systemctl enable --now openclaw-watchdog.timer
```

This checks the gateway health every 5 minutes. If it's down, it restarts automatically.

Known Bugs to Guard Against

These are active issues in the OpenClaw ecosystem. Guard against them:

1. **Baileys session state corruption:** WhatsApp sessions can corrupt after extended uptime. Mitigation: schedule WhatsApp bridge restarts every 24 hours.
2. **Cron .tmp file ghosts:** Config updates can leave orphan .tmp files that cause duplicate job execution. Mitigation: clean .tmp files after any config change.
3. **Context window overflow on multi-tool chains:** A task that calls 5+ tools in sequence can overflow context before compaction kicks in. Mitigation: set `softThresholdTokens` conservatively and use sub-agents for tool-heavy tasks.
4. **Memory search latency spikes:** Large memory directories (100+ files) can cause search timeouts. Mitigation: archive old daily logs monthly, keep active files under 50.
5. **Gateway restart during active exec:** Background processes are orphaned. Mitigation: always use session-state.md for tracking.

Part 7: Channel Setup (WhatsApp, Google, Calendar)

WhatsApp Multi-Account Configuration

WhatsApp setup in OpenClaw uses the Baileys library (open-source WhatsApp Web client). Here's the complete configuration:

```
{
  "channels": {
    "whatsapp": {
      "accounts": [
        {
          "name": "personal",
          "phone": "+1234567890",
          "baileys": {
            "authDir": "~/.openclaw/whatsapp/personal",
            "printQRInTerminal": true,
            "syncFullHistory": false,
            "browser": ["OpenClaw", "Chrome", "22.0"],
            "markOnlineOnConnect": false
          }
        },
        {
          "name": "business",
          "phone": "+0987654321",
          "baileys": {
            "authDir": "~/.openclaw/whatsapp/business",
            "printQRInTerminal": true,
            "syncFullHistory": false,
            "browser": ["OpenClaw", "Chrome", "22.0"],
            "markOnlineOnConnect": false
          }
        }
      ],
      "routing": {
        "default": "personal",
        "rules": [
          {
            "match": { "group": "Work Team" },
            "account": "business"
          }
        ]
      }
    }
  }
}
```

First-Time WhatsApp Setup

1. Start the gateway: `openclaw gateway run` (in a terminal where you can see QR codes)
2. When the QR code appears, scan it with WhatsApp on your phone
3. Wait for "Connected" confirmation in the logs
4. The auth session is saved to `authDir` — you won't need to scan again unless the session expires

WhatsApp Gotchas

- **QR codes expire fast.** You have about 30 seconds to scan. If it expires, a new one appears.
- **Multi-device limit.** WhatsApp allows up to 4 linked devices. OpenClaw counts as one.
- **History sync is expensive.** `syncFullHistory: false` is critical. Full history sync can pull thousands of messages, each consuming tokens.
- **Phone must stay online.** Your phone needs internet access. If it goes offline for 14+ days, the linked device session expires.
- **Group messages are noisy.** Consider routing rules to only respond to groups where you want the agent active.

WhatsApp Health Monitoring

Add to your HEARTBEAT.md:

```
## WhatsApp Health (check every 6 hours)
- List recent WhatsApp chats
- If the list returns empty or errors, note in error-log.md
- If no messages received in 4+ hours during active times, flag for reconnect
```

Google Workspace via gogcli

Google services (Gmail, Calendar, Drive) connect through the `gogcli` tool or direct MCP server integration.

Step-by-Step Google Setup

Step 1: Create a Google Cloud Project

1. Go to https://console.cloud.google.com/
2. Create a new project (e.g., "OpenClaw Agent")
3. Enable these APIs:
4. Gmail API
5. Google Calendar API
6. Google Drive API (if needed)

Step 2: Create OAuth 2.0 Credentials

1. Go to APIs & Services → Credentials
2. Create OAuth 2.0 Client ID
3. Application type: **Desktop application**
4. Download the JSON credentials file
5. Save as `~/.openclaw/google-credentials.json`

Step 3: Authenticate

```
# If using gogcli
gogcli auth --credentials ~/.openclaw/google-credentials.json

# This opens a browser for OAuth consent
# If on a headless server, use SSH tunnel:
ssh -L 8080:localhost:8080 your-server

# Then open the OAuth URL on your local machine
```

Step 4: Configure in openclaw.json

```
{
  "mcp": {
    "servers": {
      "google": {
        "command": "gogcli",
        "args": ["serve"],
        "env": {
          "GOOGLE_CREDENTIALS": "~/.openclaw/google-credentials.json",
          "GOOGLE_TOKEN": "~/.openclaw/google-token.json"
        }
      }
    }
  }
}
```

SSH Tunnel for Headless Auth

If your VPS doesn't have a browser:

```
# On your local machine:
ssh -L 8080:localhost:8080 user@your-vps

# Then run the auth command on the VPS
# It will give you a URL — open it in your local browser
# After consent, it redirects to localhost:8080 which tunnels to the VPS
```

Calendar Aggregation Across Accounts

If you have multiple Google accounts:

```
{
  "google": {
    "accounts": [
      {
        "name": "personal",
        "email": "you@gmail.com",
        "credentials": "~/.openclaw/google-personal-creds.json",
        "token": "~/.openclaw/google-personal-token.json"
      },
      {
        "name": "work",
        "email": "you@company.com",
        "credentials": "~/.openclaw/google-work-creds.json",
        "token": "~/.openclaw/google-work-token.json"
      }
    ]
  }
}
```

Then tell your agent in AGENTS.md:

```
## Calendar Management
- Personal calendar: you@gmail.com (default)
- Work calendar: you@company.com
- When checking schedule, ALWAYS check both calendars
- When creating events, ask which calendar unless context makes it obvious
```

Channel Routing Rules

Define how messages get routed to different agents or behaviors:

```
{
  "channels": {
    "routing": {
      "rules": [
        {
          "channel": "discord:channel:main",
          "agent": "felix",
          "behavior": "full"
        },
        {
          "channel": "discord:channel:alerts",
          "agent": "felix",
          "behavior": "notify-only"
        },
        {
          "channel": "whatsapp:personal",
          "agent": "felix",
          "behavior": "full"
        },
        {
          "channel": "whatsapp:group:*",
          "agent": "felix",
          "behavior": "mention-only"
        }
      ]
    }
  }
}
```

Behavior modes:

- `full` : Agent responds to all messages
- `mention-only` : Agent only responds when explicitly mentioned
- `notify-only` : Agent reads but only responds to urgent/actionable items
- `silent` : Agent reads but never responds (useful for monitoring)

Part 8: Production Hardening

Agent Drift Prevention

After weeks of continuous operation, agents develop "habits" — accumulated biases from conversation history that subtly shift behavior. This is normal but needs active management.

Hard Reset Every 50 Tasks

Every 50 tasks (or weekly, whichever comes first):

1. **Archive the current session**
   `Save the current conversation summary to memory/archive/YYYY-MM-DD-session.md`
2. **Curate MEMORY.md**
3. Remove outdated entries
4. Promote important lessons from daily logs
5. Keep it under 90 lines
6. Verify accuracy of all entries
7. **Review error-log.md**
8. Are the same errors still relevant?
9. Have any been fixed upstream?
10. Archive old entries
11. **Clear conversation history**
    `/compact Clear all conversation history. Start fresh. Read your workspace files.`
12. **Verify behavior**

13. Ask the agent to summarize who it is and what it's working on
14. Verify responses match expectations
15. Check that it follows SOUL.md rules

Automated Drift Detection

Add to a weekly cron job:

```
{
  "name": "drift-check",
  "schedule": "0 10 * * 5",
  "prompt": "Self-audit: Re-read SOUL.md and AGENTS.md. Compare your recent behavior (c
  "isolated": false
}
```

Security Hardening

Deny-by-Default Exec Policy

In AGENTS.md, explicitly define what's allowed:

```
## Execution Policy
### Allowed WITHOUT asking:
- Read any file in the workspace
- Write to memory/ directory
- Write to workspace files (not SOUL.md, not TOOLS.md)
- Git operations within workspace repos
- Web search and fetch
- Calendar and email read operations

### REQUIRES approval:
- Sending emails or messages
- Posting to social media
- Any destructive file operations (rm, overwrite)
- Installing packages (npm, apt)
- Running unknown scripts
- Any operation outside the workspace directory
- Creating or modifying cron jobs
- Accessing credentials or secrets
```

File Permission Lockdown

```
# Read-only for critical identity files
chmod 444 SOUL.md
chmod 444 TOOLS.md

# Agent-writable for memory and state
chmod 644 MEMORY.md
chmod 644 AGENTS.md
chmod 755 memory/

# Credentials locked down
chmod 600 ~/.openclaw/.env
chmod 600 ~/.openclaw/google-*.json
chmod 700 ~/.openclaw/whatsapp/
```

Network Security

```
# Firewall rules (ufw)
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow ssh
# Do NOT allow the OpenClaw port from anywhere
# Access via Tailscale only

sudo ufw enable
```

Backup Strategy

Daily Rsync

```
# Create backup script
cat > ~/backup-openclaw.sh << 'SCRIPT'
#!/bin/bash
BACKUP_DIR="/backups/openclaw/$(date +%Y-%m-%d)"
mkdir -p "$BACKUP_DIR"

# Backup workspace files
rsync -a ~/.openclaw/workspace-*/ "$BACKUP_DIR/workspaces/"

# Backup configs
rsync -a ~/.openclaw/openclaw.json "$BACKUP_DIR/"
rsync -a ~/.openclaw/.env "$BACKUP_DIR/"

# Backup WhatsApp auth (so you don't have to re-scan QR)
rsync -a ~/.openclaw/whatsapp/ "$BACKUP_DIR/whatsapp/"

# Backup Google tokens
rsync -a ~/.openclaw/google-*.json "$BACKUP_DIR/"

# Keep 30 days of backups
find /backups/openclaw/ -maxdepth 1 -type d -mtime +30 -exec rm -rf {} \;

echo "Backup completed: $BACKUP_DIR"
SCRIPT

chmod +x ~/backup-openclaw.sh

# Add to crontab
(crontab -l 2>/dev/null; echo "0 3 * * * /home/openclaw/backup-openclaw.sh") | crontab
```

Off-Site Backup

For critical setups, push backups to a remote location:

```
# After local backup, push to remote
rsync -az /backups/openclaw/ remote-server:/backups/openclaw/

# Or use rclone for cloud storage
rclone sync /backups/openclaw/ b2:my-bucket/openclaw-backups/
```

Compaction Defense: 4 Layers

Compaction is necessary (you can't have infinite context), but uncontrolled compaction destroys your agent's working memory. Here are four layers of defense:

Layer 1: session-state.md (Real-Time)

Always maintain a `session-state.md` with current task state. Updated after every significant action.

```
# Session State — 2026-02-24 14:30

## Current Task
Building pricing page for [project]. Currently integrating Stripe checkout.

## Last Completed Step
Created components/PricingCard.tsx with three tier options.

## Next Step
Add Stripe checkout session creation to pages/api/checkout.ts

## Key State
- Stripe test key: in .env.local
- Product IDs: basic=prod_xxx, pro=prod_yyy, enterprise=prod_zzz
- Using Next.js 14 App Router
```

Layer 2: memoryFlush (On Compaction)

Extracts key information before compaction destroys it.

```
{
  "memoryFlush": {
    "enabled": true,
    "softThresholdTokens": 40000,
    "prompt": "Extract: decisions, state changes, lessons, blockers, corrections."
  }
}
```

Layer 3: active-tasks.md (Persistent)

For multi-step projects, maintain an active-tasks.md that survives across sessions.

Layer 4: MEMORY.md Curation (Weekly)

Regular human-assisted curation of long-term memory. The agent proposes updates, you approve.

The Digestive Pipeline

Think of your agent's information flow like a digestive system:

```
Input (messages, tool outputs, research)
    ↓
Working Memory (conversation context) — 50K-100K tokens
    ↓
Compaction (memoryFlush extracts, rest is summarized)
    ↓
Daily Log (memory/YYYY-MM-DD.md) — raw capture
    ↓
Curation (weekly review)
    ↓
Long-Term Memory (MEMORY.md) — distilled wisdom
    ↓
Archive (monthly cleanup of old daily logs)
```

Each stage reduces volume and increases signal-to-noise ratio. The key insight: **information should flow downward automatically, but curation should involve human review.** The agent can propose MEMORY.md updates, but you should approve significant changes.

Part 9: Copy-Paste Configs

Complete openclaw.json

```
{
  "gateway": {
    "port": 19847,
    "host": "127.0.0.1",
    "auth": {
      "token": "GENERATE_WITH_openssl_rand_hex_32"
    }
  },

  "model": "claude-sonnet-4-20250514",

  "memory": {
    "enabled": true,
    "memoryFlush": {
      "enabled": true,
      "softThresholdTokens": 40000,
      "prompt": "Extract and save: 1) Decisions and reasoning, 2) State changes, 3) Les
    },
    "search": {
      "enabled": true,
      "hybridSearch": {
        "vectorWeight": 0.7,
        "textWeight": 0.3,
        "candidateMultiplier": 4
      },
      "embedding": {
        "provider": "openai",
        "model": "text-embedding-3-small"
      }
    }
  },

  "heartbeat": {
    "intervalMs": 7200000,
    "prompt": "Read HEARTBEAT.md if it exists. Follow it strictly. If nothing needs att
  },

  "cron": {
    "jobs": [
      {
        "name": "morning-briefing",
```

```
        "schedule": "0 8 * * 0-4",
        "prompt": "Morning briefing: 1) Today's calendar events (all accounts), 2) Urge
        "isolated": true,
        "deliverTo": "discord:channel:YOUR_CHANNEL_ID",
        "model": "claude-sonnet-4-20250514"
      },
      {
        "name": "memory-maintenance",
        "schedule": "0 22 * * *",
        "prompt": "Memory maintenance: 1) Review today's daily log, 2) Update MEMORY.md
        "isolated": false
      },
      {
        "name": "weekly-review",
        "schedule": "0 9 * * 5",
        "prompt": "Weekly review: Read all daily logs from this week. Summarize: accomp
        "isolated": true,
        "deliverTo": "discord:channel:YOUR_CHANNEL_ID",
        "model": "claude-sonnet-4-20250514"
      }
    ]
  },

  "channels": {
    "discord": {
      "enabled": true,
      "token": "YOUR_DISCORD_BOT_TOKEN",
      "channels": ["YOUR_CHANNEL_ID"]
    }
  }
}
```

Memory Configuration (Standalone)

If you want to add memory to an existing setup:

```
{
  "memory": {
    "enabled": true,
    "memoryFlush": {
      "enabled": true,
      "softThresholdTokens": 40000,
      "prompt": "Extract and save: 1) Decisions and reasoning, 2) State changes, 3) Les
    },
    "search": {
      "enabled": true,
      "hybridSearch": {
        "vectorWeight": 0.7,
        "textWeight": 0.3,
        "candidateMultiplier": 4
      },
      "embedding": {
        "provider": "openai",
        "model": "text-embedding-3-small"
      }
    }
  }
}
```

Don't forget:

1. `export OPENAI_API_KEY=sk-...` in your environment
2. Create `MEMORY.md` manually
3. Create `memory/` directory
4. Add memory protocols to AGENTS.md

Heartbeat Config

```
{
  "heartbeat": {
    "intervalMs": 7200000,
    "prompt": "Read HEARTBEAT.md if it exists. Follow it strictly. If nothing needs att
  }
}
```

HEARTBEAT.md Template

```
# HEARTBEAT.md — Periodic Checks

## Priority Checks (every heartbeat)
- Any unread urgent emails? (starred or from key contacts)
- Calendar events in the next 2 hours?

## Regular Checks (rotate through these)
- [ ] Email inbox — any actionable items?
- [ ] GitHub notifications — any PRs or issues needing attention?
- [ ] WhatsApp — connection healthy?
- [ ] Weather — relevant if Sam might go out?

## Check Tracking
Update memory/heartbeat-state.json after each check cycle:
{
  "lastChecks": {
    "email": "timestamp",
    "calendar": "timestamp",
    "github": "timestamp",
    "whatsapp": "timestamp",
    "weather": "timestamp"
  }
}

## Quiet Hours
- 23:00-08:00: HEARTBEAT_OK unless truly urgent
- During known busy periods: HEARTBEAT_OK
```

Watchdog systemd Timer

/etc/systemd/system/openclaw-watchdog.service

```
[Unit]
Description=OpenClaw Gateway Watchdog
After=network.target

[Service]
Type=oneshot
User=openclaw
ExecStart=/bin/bash -c 'if ! curl -sf http://127.0.0.1:19847/health > /dev/null 2>&1; t
```

/etc/systemd/system/openclaw-watchdog.timer

```
[Unit]
Description=Check OpenClaw Gateway Health Every 5 Minutes

[Timer]
OnBootSec=60
OnUnitActiveSec=300
AccuracySec=30

[Install]
WantedBy=timers.target
```

Enable

```
sudo systemctl daemon-reload
sudo systemctl enable --now openclaw-watchdog.timer

# Verify
sudo systemctl list-timers | grep openclaw
```

Error Log Auto-Capture for AGENTS.md

Add this block to your AGENTS.md:

```
## Error Logging Protocol

### When to Log
Append to `memory/error-log.md` when ANY of these occur:
- A tool call returns an error or unexpected result
- The user corrects your behavior or output
- You discover a workaround for a known issue
- You make a mistake and realize it
- An API rate limit is hit
- A service is temporarily unavailable

### Log Format
## [YYYY-MM-DD HH:MM] [Category: Tool/API/Behavior/Config]
**What happened:** [one line description]
**Why:** [root cause, one line]
**Fix/Workaround:** [what resolved it, one line]
**Prevention:** [how to avoid in future, one line]

### Review Protocol
- Check error-log.md before attempting operations similar to past errors
- During weekly memory maintenance, review for patterns
- Promote recurring issues to MEMORY.md as permanent lessons
```

Gateway systemd Service

For production VPS deployments:

/etc/systemd/system/openclaw-gateway.service

```
[Unit]
Description=OpenClaw Gateway
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
User=openclaw
WorkingDirectory=/home/openclaw/.openclaw
ExecStart=/usr/local/bin/openclaw gateway run
Restart=always
RestartSec=10
StandardOutput=append:/var/log/openclaw/gateway.log
StandardError=append:/var/log/openclaw/gateway-error.log

# Security hardening
NoNewPrivileges=true
ProtectHome=read-only
ReadWritePaths=/home/openclaw/.openclaw

# Environment
EnvironmentFile=/home/openclaw/.openclaw/.env

[Install]
WantedBy=multi-user.target
```

Setup

```
# Create log directory
sudo mkdir -p /var/log/openclaw
sudo chown openclaw:openclaw /var/log/openclaw

# Enable and start
sudo systemctl daemon-reload
sudo systemctl enable openclaw-gateway
sudo systemctl start openclaw-gateway

# Check status
sudo systemctl status openclaw-gateway
```

Part 10: Troubleshooting Decision Tree

"My Agent Doesn't Remember Anything"

```
Agent has amnesia
├── Is MEMORY.md created?
│   ├── NO → Create it manually. See Part 2 template.
│   └── YES → Continue
├── Is memoryFlush enabled?
│   ├── NO → Add memoryFlush config to openclaw.json. See Part 3.
│   └── YES → Continue
├── Does AGENTS.md have boot sequence?
│   ├── NO → Add "Read MEMORY.md" to boot sequence. See Part 2.
│   └── YES → Continue
├── Is memory search working?
│   ├── Check: Does OPENAI_API_KEY exist in environment?
│   │   ├── NO → Set it. Memory search needs embeddings.
│   │   └── YES → Continue
│   ├── Check: Are there files in memory/ directory?
│   │   ├── NO → Agent isn't writing daily logs. Fix AGENTS.md.
│   │   └── YES → Continue
│   └── Check: Is hybridSearch configured?
│       ├── NO → Add hybrid search config. See Part 3.
│       └── YES → Memory system is configured. Issue may be:
│           ├── MEMORY.md is too bloated (>90 lines) → Curate it
│           ├── Daily logs are too sparse → Improve memoryFlush prompt
│           └── Agent isn't reading memory on boot → Verify AGENTS.md
```

"WhatsApp Keeps Disconnecting"

```
WhatsApp disconnects
├── Is the phone online with internet?
│   ├── NO → Phone must stay online for linked devices.
│   └── YES → Continue
├── Has it been >14 days since linking?
│   ├── POSSIBLE → Re-link: delete auth dir, restart, scan new QR.
│   └── NO → Continue
├── Are there reconnect loops in logs?
│   ├── YES → Known Baileys issue.
│   │   ├── Delete auth session: rm -rf ~/.openclaw/whatsapp/personal/
│   │   ├── Restart gateway: openclaw gateway restart
│   │   ├── Scan new QR code
│   │   └── Add WhatsApp health check to heartbeat/cron
│   └── NO → Continue
├── Is it a specific group/contact?
│   ├── YES → May be blocked or group settings issue
│   └── NO → General connection issue
│       ├── Check server resources: free -m, df -h
│       ├── Check network: ping api.whatsapp.com
│       └── Check Node.js version: node --version (need 18+)
```

"My Token Costs Are Too High"

```
Tokens burning too fast
├── Check: How many MCP servers are connected?
│   ├── >3 → Disconnect unused servers. Each adds 2-5K tokens/call.
│   └── ≤3 → Continue
├── Check: How many skills installed?
│   ├── >35 → Remove unused skills. Each adds 500-2K tokens/call.
│   └── ≤35 → Continue
├── Check: Heartbeat frequency?
│   ├── <1 hour → Increase to 2-4 hours. Each heartbeat costs 3-8K tokens.
│   └── ≥1 hour → Continue
├── Check: How many cron jobs?
│   ├── >5 → Audit. Each cron run costs 5-15K tokens.
│   └── ≤5 → Continue
├── Check: Tool output sizes?
│   ├── Doing bulk email/web fetches? → Switch to metadata-first approach
│   ├── Fetching large web pages? → Use sub-agents for research
│   └── Normal tool use → Continue
├── Check: Conversation history length?
│   ├── Not compacting? → Lower softThresholdTokens to 30000
│   ├── Long sessions without /compact? → Compact after heavy tasks
│   └── Already compact → Continue
├── Check: Model?
│   ├── Using Opus for everything? → Switch routine tasks to Sonnet
│   └── Already using Sonnet → You may genuinely need this many tokens
│       └── Consider sub-agents to isolate expensive contexts
```

"Cron Jobs Don't Deliver"

```
Cron jobs run but nothing appears
├── Is the job isolated?
│   ├── YES → Is deliverTo set?
│   │   ├── NO → Add deliverTo with your channel ID. This is the #1 cause.
│   │   └── YES → Is the channel ID correct?
│   │       ├── Check by sending a test message to that channel manually
│   │       └── If channel ID is wrong → Fix it
│   └── NO (main session) → Is the main session connected to a channel?
│       ├── NO → Main session needs an active channel connection
│       └── YES → Continue
├── Is the schedule correct?
│   ├── Check timezone: cron uses server timezone
│   ├── Test with "* * * * *" (every minute) to verify it fires
│   └── Check: crontab syntax matches your intent
├── Is the gateway running?
│   ├── NO → Start it. Cron needs the gateway.
│   └── YES → Continue
├── Check for .tmp ghost files:
│   └── find ~/.openclaw/ -name "*.tmp" -type f
│       ├── Found files → Delete them, restart gateway
│       └── No files → Continue
├── Check gateway logs for errors:
│   └── tail -100 /var/log/openclaw/gateway.log | grep -i cron
│       ├── Errors found → Address specific error
│       └── No errors → Job may be running but output is empty
│           └── Test the prompt manually to verify it produces output
```

"Agent Is Slow to Respond"

```
Agent takes a long time to respond
├── Check: How large is the context window?
│   ├── >80K tokens → Compact immediately. Large context = slow.
│   └── <80K → Continue
├── Check: How many tool calls per response?
│   ├── >5 → Agent is over-tooling. Simplify the task or use sub-agents.
│   └── ≤5 → Continue
├── Check: API latency
│   ├── Are all LLM calls slow? → Anthropic may be experiencing load.
│   │   └── Check status.anthropic.com
│   └── Only some calls slow → Continue
├── Check: Memory search latency
│   ├── Many files in memory/? → Archive old logs (keep 14 days max)
│   ├── Using QMD? → QMD is 2-5x slower than default. Expected.
│   └── Normal file count → Continue
├── Check: Server resources
│   ├── CPU: top → is Node.js pegging CPU?
│   ├── RAM: free -m → less than 1GB free?
│   ├── Disk: df -h → less than 10% free?
│   └── Network: ping api.anthropic.com
├── Check: Gateway health
│   └── curl http://127.0.0.1:19847/health
│       ├── Errors → Restart gateway
│       └── Healthy → The bottleneck is API-side. Normal for complex tasks.
```

"Agent Keeps Making the Same Mistake"

```
Agent repeats errors
├── Does memory/error-log.md exist?
│   ├── NO → Create it. Add error logging protocol to AGENTS.md.
│   └── YES → Is the error logged?
│       ├── NO → Agent isn't logging errors. Reinforce the protocol.
│       └── YES → Agent isn't reading error-log.md before similar operations.
│           ├── Add to AGENTS.md: "Check error-log.md before [specific operation]"
│           └── Promote the error to MEMORY.md for higher visibility
├── Is the agent being corrected?
│   ├── YES → Is the correction being saved?
│   │   ├── NO → "Remember this correction" → write to MEMORY.md
│   │   └── YES → May be a compaction issue (correction gets compacted out)
│   │       └── Write the correction to SOUL.md as a permanent behavioral rule
│   └── NO → Correct explicitly: "Don't do X because Y. Save this rule."
```

Appendix A: Glossary

Agent: An AI assistant running on OpenClaw. Has its own workspace, memory, and personality.

AGENTS.md: The operating manual for your agent. Defines boot sequence, protocols, and rules.

Baileys: Open-source library for connecting to WhatsApp Web. Used by OpenClaw for WhatsApp integration.

Channel: A communication endpoint — Discord channel, WhatsApp chat, etc.

Compaction: The process of summarizing older conversation history to free context window space.

Context Window: The total amount of text (in tokens) a language model can "see" at once. Currently 200K for Claude.

Cron: Scheduled tasks that run on a time-based schedule (like Unix cron jobs).

Gateway: The OpenClaw server process that orchestrates agents, channels, and tools.

Heartbeat: A periodic poll that gives the agent a chance to check for new work or do background tasks.

MCP (Model Context Protocol): A standard for connecting AI models to external tools and data sources.

memoryFlush: The system that extracts important information before conversation history is compacted.

MEMORY.md: The curated long-term memory file. Created and maintained manually.

Session: A conversation context with accumulated history. Persists until compacted or reset.

SOUL.md: Defines the agent's personality, values, and behavioral rules.

Sub-agent: A temporary agent spawned for a specific task. Runs in its own context, reports back, and terminates.

Token: A unit of text (~4 characters in English). Used for billing and context window measurement.

USER.md: Defines who the agent is helping — name, preferences, work context.

Appendix B: Cost Calculator

Monthly Cost Estimation

```
Daily cost = (Messages × Avg tokens per message × Price per token)
           + (Heartbeats per day × Avg tokens per heartbeat × Price per token)
           + (Cron jobs per day × Avg tokens per cron × Price per token)

Monthly cost = Daily cost × 30
```

Quick Reference Prices (as of Feb 2026)

Model	Input (per 1M tokens)	Output (per 1M tokens)
Claude Opus 4	$15.00	$75.00
Claude Sonnet 4	$3.00	$15.00

Typical Scenarios

Scenario A: Personal assistant, light use
- Model: Sonnet
- 10 messages/day × 5K tokens = 50K tokens
- 6 heartbeats/day × 4K tokens = 24K tokens
- 2 cron jobs/day × 8K tokens = 16K tokens
- Daily: ~90K input + ~30K output ≈ **$0.72/day = $22/month**

Scenario B: Work assistant, moderate use
- Model: Sonnet (with occasional Opus for complex tasks)
- 30 messages/day × 8K tokens = 240K tokens
- 12 heartbeats/day × 5K tokens = 60K tokens

- 4 cron jobs/day × 10K tokens = 40K tokens
- Daily: ~340K input + ~100K output ≈ **$2.52/day = $76/month**

Scenario C: Autonomous builder, heavy use
- Model: Opus coordinator + Sonnet workers
- 50 messages/day × 15K tokens (Opus) = 750K tokens @ Opus rate
- 20 sub-agent tasks/day × 20K tokens (Sonnet) = 400K tokens @ Sonnet rate
- Heartbeats + cron: ~200K tokens @ Sonnet rate
- Daily: **$15-25/day = $450-750/month**

Appendix C: The First 7 Days Checklist

Day 1: Foundation

- [ ] Server set up (Hetzner CX43 or equivalent)
- [ ] OpenClaw installed
- [ ] API keys configured (.env file)
- [ ] openclaw.json created with security settings
- [ ] SOUL.md written (under 800 tokens)
- [ ] USER.md written (under 500 tokens)
- [ ] AGENTS.md created with boot sequence
- [ ] MEMORY.md created with initial context
- [ ] memory/ directory created
- [ ] Gateway started successfully
- [ ] First conversation works

Day 2: Memory

- [ ] memoryFlush enabled and tested
- [ ] Memory search working (no 401 errors)
- [ ] Agent creates daily log file
- [ ] Agent reads MEMORY.md on boot
- [ ] error-log.md created
- [ ] Compaction tested (long conversation → verify memoryFlush fires)

Day 3: Channels

- [ ] Primary channel connected (Discord/WhatsApp)

- [ ] Channel routing configured
- [ ] Test: send message via channel, get response
- [ ] Test: agent sends proactive message to channel

Day 4: Automation

- [ ] Heartbeat configured and tested
- [ ] HEARTBEAT.md created
- [ ] First cron job created and verified
- [ ] Cron delivery confirmed (message appears in channel)

Day 5: Tools

- [ ] Google accounts connected
- [ ] Calendar access verified
- [ ] Email access verified
- [ ] GitHub access verified (if needed)
- [ ] TOOLS.md updated with all available tools

Day 6: Hardening

- [ ] Watchdog timer installed
- [ ] Backup script created and tested
- [ ] File permissions locked down
- [ ] session-state.md protocol verified
- [ ] Tailscale configured (if using remote access)

Day 7: Validation

- [ ] Full autonomous task: morning briefing fires and delivers
- [ ] Memory test: restart session, verify agent remembers yesterday

- [ ] Recovery test: simulate compaction, verify session-state.md works
- [ ] Error handling: trigger a known error, verify error-log.md capture
- [ ] Cost check: review API usage, adjust if over budget

Final Words

OpenClaw is the most powerful AI agent framework available today. It's also the most unforgiving. The gap between "installed" and "useful" is wider than any documentation suggests.

But once you cross that gap — once your agent remembers your preferences, monitors your inbox, builds while you sleep, and catches its own mistakes — you'll wonder how you ever worked without it.

The configs in this guide are a starting point. Every setup is different. You'll tweak, break, fix, and improve. That's the process. The key is to start with a solid foundation (which you now have) and iterate from there.

Three rules for the long term:

1. **Curate relentlessly.** Memory is your agent's most valuable asset. Treat it like a garden, not a landfill.
2. **Reset regularly.** Agent drift is inevitable. Hard resets are maintenance, not failure.
3. **Budget consciously.** Tokens are real money. Know where yours go. Optimize deliberately.

Welcome to the future of personal computing. It's messy, expensive, and occasionally maddening. It's also genuinely transformative once you get it right.

Now go build something.

The OpenClaw Field Manual — Version 1.0
Written for the community, by the community.
Last updated: February 2026

License: This guide is sold as a digital product. Personal use only. Do not redistribute.
Support: Questions? Issues? Find us in the OpenClaw community Discord.
Updates: Major version updates are free for all purchasers.