OpenClaw Deployment Guide

OpenClaw Deployment Guide

Secure Setup on a Dedicated Mac mini

Version: 1.6
Date: February 2026
Audience: Semi-technical users setting up a personal AI assistant

---

> **Disclaimer:** This guide is provided for educational purposes only. The authors make no warranties about the completeness, reliability, or security of this information. Implement at your own risk. This is not professional security consulting, and you should consult a qualified security professional for your specific environment and threat model. Security requirements vary by jurisdiction and use case.

Table of Contents

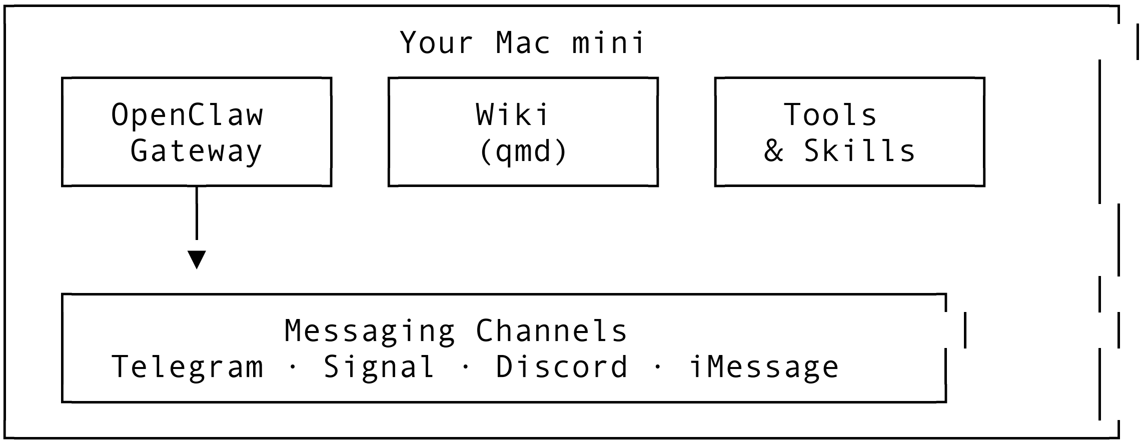
Introduction

OpenClaw is a self-hosted AI assistant framework that connects frontier AI models (like Claude) to your messaging apps, files, and tools. Unlike cloud-based assistants, OpenClaw runs on your own hardware, giving you:

- **Privacy**: Your conversations stay on your machine
- **Control**: You decide what the AI can access
- **Customization**: Configure personality, tools, and integrations
- **Persistence**: The AI remembers context across sessions

This guide walks you through setting up OpenClaw securely on a dedicated Mac mini.

What You'll Build



Prerequisites

Hardware

- Mac mini (M1 or newer recommended)
- 16GB+ RAM recommended
- 256GB+ storage
- Stable internet connection

Accounts You'll Need

- **Claude Pro or Max subscription** ($20-100/month) — https://claude.ai
- **Telegram account** (recommended starting channel)
- Apple ID (for the Mac)

> **Cost Tip:** A Claude Pro ($20/mo) or Max ($100/mo) subscription is significantly cheaper than direct API access for personal use. The subscription includes API access via OAuth authentication.

Time Required

- Initial setup: 2-3 hours
- Security hardening: 30 minutes
- Wiki setup: 30 minutes

Phase 1: Create Your AI's Identity

Why? Keeping your AI's accounts separate from your personal accounts provides security isolation and makes management easier. Your AI needs both external identity (accounts) and internal identity (personality files).

External Identity (Accounts)

1.1 Create a Dedicated Email

Create a new email address specifically for your AI assistant:

```
Examples:
- yourname-ai@proton.me
- assistant.yourname@gmail.com
- jarvis@yourdomain.com
```

Recommended providers: - ProtonMail (privacy-focused) - Gmail (convenient, good 2FA) - Custom domain (professional)

1.2 Create a Password Manager Entry

Store all AI-related credentials in a dedicated password manager folder:

```
📁 AI Assistant
  ├── Email credentials
  ├── Claude subscription (claude.ai login)
  ├── Telegram Bot token
  ├── Mac mini login
  └── Any other services
```

Internal Identity (Personality Files)

Your AI's personality and behavior are defined by files in its workspace:

SOUL.md — Who Your AI Is

This file defines your AI's core personality, tone, and values.

Location: `~/.openclaw/workspace/SOUL.md`

```
# SOUL.md - Who You Are

## Core Truths
- Be genuinely helpful, not performatively helpful
- Have opinions — an assistant with no personality is just a search engine
- Be resourceful before asking — try to figure it out first
- Earn trust through competence

## Boundaries
- Private things stay private
- When in doubt, ask before acting externally
- You're not the user's voice — be careful in group chats

## Vibe
```

```
Be the assistant you'd actually want to talk to.
Concise when needed, thorough when it matters.
```

Customize this to match how you want your AI to behave. Want it formal? Casual? Funny? This is where you define that.

AGENTS.md — Operating Instructions

This file tells your AI how to operate within its workspace.

Location: `~/.openclaw/workspace/AGENTS.md`

Key sections to include: - **Memory management** — How to use daily notes and long-term memory - **Safety rules** — What requires permission vs. what's autonomous - **Tool usage** — How to use specific integrations - **Wiki workflow** — When to document research and decisions

USER.md — About You

This file helps your AI understand who it's helping.

Location: `~/.openclaw/workspace/USER.md`

```
# USER.md - About Your Human

- **Name:** [Your name]
- **Location:** [City/timezone]
- **Interests:** [Key interests]
- **Communication style:** [How you like to communicate]
```

MEMORY.md — Long-term Memory

Your AI wakes up fresh each session. This file is its persistent memory.

Location: `~/.openclaw/workspace/MEMORY.md`

The AI reads this at session start and updates it with important information to remember across sessions.

1.3 Set Up Claude Pro/Max Subscription (Recommended)

Using a Claude subscription is **significantly cheaper** than direct API access for personal use.

Option A: Claude Pro ($20/month) — Good for moderate use **Option B: Claude Max ($100/month)** — Higher limits, best value for heavy use

1. Go to https://claude.ai
2. Sign up using your AI's dedicated email
3. Subscribe to Pro or Max plan
4. Install the Claude CLI (we'll do this in Phase 3)

The Claude CLI authenticates via OAuth with your subscription — no separate API key needed.

Cost comparison: | Method | Typical Monthly Cost | |——|————————| | Claude Pro subscription | $20 flat | | Claude Max subscription | $100 flat | | Direct API (pay-as-you-go) | $50-500+ variable |

1.3b Alternative: Direct API Access

If you prefer pay-as-you-go or need higher limits:

1. Go to https://console.anthropic.com
2. Sign up using your AI's email
3. Navigate to API Keys
4. Create a new key, name it "OpenClaw - Mac mini"
5. Save the key securely

Note: Direct API is billed per token. Heavy use can get expensive quickly.

1.4 Create a Telegram Bot

1. Open Telegram and message @BotFather
2. Send `/newbot`
3. Choose a name (e.g., "Your AI Assistant")
4. Choose a username (must end in `bot`, e.g., `yourname_ai_bot`)
5. Save the bot token — you'll need it later

Phase 2: Prepare the Mac mini

2.1 Initial macOS Setup

1. **Create a dedicated user account** for the AI:

   - System Settings → Users & Groups → Add User
   - Username: `assistant` (or your AI's name)
   - Make it an Administrator account

2. **Log in as the new user** and complete setup

3. **Enable Remote Access** (optional, for headless operation):

   - System Settings → General → Sharing
   - Enable "Remote Login" (SSH)
   - Enable "Screen Sharing" (VNC)

2.2 Install Homebrew

Open Terminal and run:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/
        Homebrew/install/HEAD/install.sh)"
```

Follow the post-install instructions to add Homebrew to your PATH.

2.3 Install Node.js

```
brew install node@22
```

Verify installation:

```
node --version  # Should show v22.x.x
npm --version   # Should show 10.x.x
```

Phase 3: Install OpenClaw

3.1 Install OpenClaw

```
npm install -g openclaw
```

3.2 Install and Configure Claude CLI

If using Claude Pro/Max subscription (recommended):

```
# Install Claude CLI
brew install claude

# Authenticate with your Claude account
claude login
```

This opens a browser to authenticate with your Claude subscription via OAuth. Once authenticated, OpenClaw can use your subscription.

3.3 Run the Setup Wizard

```
openclaw doctor
```

The wizard will guide you through: 1. **Authentication** — It will detect your Claude CLI setup automatically 2. **Model selection** — Recommend Claude Opus 4.5 for best security 3. **Initial workspace setup**

If using direct API instead, the wizard will prompt for your API key.

3.3 Verify Installation

```
openclaw status
```

You should see the Gateway running and your agent configured.

3.4 Start the Gateway Service

```
openclaw gateway start
```

This installs a LaunchAgent that keeps OpenClaw running in the background.

Phase 4: Security Hardening

This is critical. An AI with shell access is powerful — secure it properly.

Option A: Ask Your Agent to Harden Security

Once OpenClaw is running, you can ask:

> "Review the OpenClaw security documentation and harden our setup. Move secrets to a .env file, add trusted proxies for nginx, set up plugin allowlists, and run a security audit. Show me what you're changing before you do it."

Your AI will review docs, make recommendations, and implement changes with your approval.

Option B: Manual Security Setup

4.1 Run the Security Audit

```
openclaw security audit
```

Address any issues flagged as "critical" or "warn".

4.2 Configure DM Policy

By default, OpenClaw uses "pairing" mode — unknown senders must be approved. Verify this is set:

```
openclaw config get channels.telegram.dmPolicy
# Should return: "pairing"
```

4.3 Move Secrets to Environment File

Create a secure `.env` file:

```
# Set restrictive permissions first
touch ~/.openclaw/.env
chmod 600 ~/.openclaw/.env
```

Edit `~/.openclaw/.env`:

```
# OpenClaw Secrets - DO NOT COMMIT TO GIT
TELEGRAM_BOT_TOKEN=your_bot_token_here

# Only needed if using direct API (not needed for Claude Pro/Max
        subscription)
# ANTHROPIC_API_KEY=your_api_key_here
```

4.4 Configure Trusted Proxies (if using reverse proxy)

If you plan to expose OpenClaw through nginx or Caddy:

```
openclaw config set gateway.trustedProxies '["127.0.0.1"]'
```

4.5 Set Plugin Allowlist

Only enable plugins you explicitly need:

```
openclaw config set plugins.allow '["telegram"]'
```

4.6 Verify File Permissions

```
ls -la ~/.openclaw/
# Directory should be drwx------ (700)

ls -la ~/.openclaw/openclaw.json
# Config should be -rw------- (600)
```

4.7 Security Best Practices

Do	Don't
Use pairing mode for DMs	Set dmPolicy to "open"
Use allowlists for groups	Allow all groups
Keep plugins minimal	Install untrusted plugins
Use Claude Opus 4.5	Use smaller models for tool-enabled bots
Review audit regularly	Ignore security warnings

> **Note on Prompt Injection:** While Claude Opus 4.5 demonstrates stronger prompt injection resistance than smaller models in benchmarks, no LLM is immune to all adversarial prompts. Defense in depth (network isolation, allowlists, guardrails, monitoring) remains essential. Never rely solely on model-level safety for critical security boundaries.

Phase 5: Connect Messaging Channels

5.1 Configure Telegram

Edit your config to add the Telegram bot:

```
openclaw config set channels.telegram.enabled true
openclaw config set channels.telegram.botToken '${TELEGRAM_BOT_TOKEN}'
openclaw config set channels.telegram.dmPolicy 'pairing'
```

Restart the gateway:

```
openclaw gateway restart
```

5.2 Pair Your Account

1. Open Telegram and message your bot
2. The bot will respond with a pairing code
3. On your Mac mini, run:

```
openclaw pairing list telegram
openclaw pairing approve telegram <CODE>
```

1. You're now connected!

5.3 Optional: Add More Channels

OpenClaw supports: - **Signal** — Private messaging - **Discord** — Server communities - **WhatsApp** — Requires linked device - **iMessage** — macOS only, requires permissions

See https://docs.openclaw.ai for channel-specific setup.

Phase 6: Set Up the Knowledge Wiki

A personal wiki lets your AI document research, decisions, and ideas — building institutional knowledge over time.

Option A: Ask Your Agent to Set It Up

Once OpenClaw is running, you can simply ask:

> "Set up a knowledge wiki for us using qmd. Create folders for research, ideas, decisions, projects, and meetings. Update AGENTS.md so you know to document our discussions there."

Your AI will handle the installation and configuration. Skip to section 6.6 to understand how it works.

Option B: Manual Setup

6.1 Install qmd

qmd is a local semantic search engine for markdown files:

```
brew install oven-sh/bun/bun
bun install -g https://github.com/tobi/qmd
```

Add to your PATH:

```
echo 'export PATH="$HOME/.bun/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc
```

6.2 Create Wiki Structure

```
mkdir -p ~/.openclaw/workspace/wiki/
        {research,ideas,decisions,projects,meetings}
```

6.3 Create Wiki Index

Create ~/.openclaw/workspace/wiki/README.md:

```
# Knowledge Wiki

## Structure
- **research/** — Deep dives and investigations
- **ideas/** — Brainstorms and explorations
- **decisions/** — What we decided and why (ADRs)
- **projects/** — Active project documentation
- **meetings/** — Discussion summaries

## Search Commands
qmd search "keyword"      # Fast keyword search
qmd vsearch "concept"     # Semantic search
qmd query "question"      # Best quality (hybrid + AI)
```

6.4 Index the Wiki

```
qmd collection add ~/.openclaw/workspace/wiki --name wiki
qmd context add "qmd://wiki" "Personal knowledge base"
qmd embed
```

6.5 Update Your AI's Instructions

Add to your workspace's AGENTS.md:

```
## Wiki Documentation

When we discuss ideas, research, or make decisions:
- Document research in `wiki/research/YYYY-MM-DD-topic.md`
- Capture ideas in `wiki/ideas/`
- Record decisions in `wiki/decisions/` (ADR format)

Search with: qmd search/vsearch/query
Reindex with: qmd update && qmd embed
```

How It Works

Your AI can now: 1. **Write** research notes and decisions to the wiki 2. **Search** past knowledge using semantic search 3. **Build** institutional memory over time

Example conversation: > You: “Research the best home automation protocols” > AI: *researches and writes findings to wiki/research/2026-02-02-home-automation.md* > > Later… > You: “What did we learn about home automation?” > AI: *searches wiki and retrieves the research*

Phase 7: Optimize Token Usage

Even with a flat-rate subscription, optimizing token usage improves response quality and avoids hitting rate limits.

7.1 Subscription vs API Pricing

If using Claude Pro/Max subscription (recommended): - Fixed monthly cost ($20-100) - Rate limits apply but no per-token charges - Optimization helps avoid rate limit throttling

If using direct API (pay-as-you-go):

Model	Input	Output	Best For
Claude Opus 4.5	$15/M	$75/M	Complex tasks, security
Claude Sonnet 4.5	$3/M	$15/M	General use

Direct API costs can add up quickly — subscription is usually better value for personal use.

7.2 Configure Compaction

Compaction summarizes old context to reduce tokens:

```
openclaw config set agents.defaults.compaction.mode 'safeguard'
```

7.3 Use Memory Efficiently

OpenClaw’s memory system: - **Session memory**: Current conversation - **Long-term memory**: `MEMORY.md` file - **Daily notes**: `memory/YYYY-MM-DD.md`

Teach your AI to write important things to files rather than relying on context.

7.4 Set Up Heartbeats (Instead of Always-On)

Instead of constant polling, use heartbeats for periodic check-ins:

```json
{
  "heartbeat": {
    "every": "30m",
    "activeHours": {
      "start": "08:00",
      "end": "22:00"
    }
  }
}
```

7.5 Monitor Usage

For Claude Pro/Max subscription: - Check usage at https://claude.ai/settings - Monitor for rate limit warnings - Upgrade to Max if hitting Pro limits frequently

For direct API: - Check dashboard at https://console.anthropic.com/dashboard - Set up billing alerts to avoid surprises - Review usage patterns monthly

---

Phase 8: Implement Guardrails

Guardrails are safety boundaries that control what your AI can and cannot do autonomously. Think of them as permission levels for different actions.

> **Note:** Guardrail systems are still being developed in the community. The concepts below represent best practices that are actively being implemented.

Core Guardrail Categories

8.1 Isolation Guardrails

Control what systems and data your AI can access.

Resource	Full Access	Read-Only	No Access
Filesystem	Workspace only	Specific folders	System files
Network	Internal only	Specific domains	Public internet
Messaging	Approved contacts	—	Unknown senders
Databases	Personal DBs	—	Production systems

Implementation approaches: - Use OpenClaw’s sandbox mode for exec isolation - Configure filesystem access in AGENTS.md - Use channel allowlists for messaging isolation

8.2 Behavior Guardrails

Control what actions your AI can take autonomously vs. with approval.

Action Type	Autonomous	Draft/Propose	Requires Approval
Read files	✅	—	—
Write to workspace	✅	—	—
Web search	✅	—	—
Send messages (known contacts)	—	✅ Draft first	—
Send messages (new contacts)	—	—	✅
Send emails	—	✅ Draft only	Send requires approval
Social media posts	—	✅ Draft only	Post requires approval
Financial transactions	—	—	✅ Always
Delete files	—	—	✅

Example: Email Guardrail

When you give your AI email access, you might want to: - ✅ Read incoming emails autonomously - ✅ Draft reply suggestions - ❌ Actually send emails without your approval

Configure this in AGENTS.md:

```
## Email Rules
- You may read and summarize emails freely
- You may DRAFT replies, but save them as drafts only
- NEVER send emails without explicit approval
- Present drafts for review before sending
```

8.3 Approval Guardrails

Define what requires explicit human approval before execution.

Always require approval for: - Anything that leaves the local machine (external APIs, posts, sends) - Destructive operations (delete, overwrite) - Financial or legal actions - Actions affecting other people - Anything you're uncertain about

Approval workflow:

```
AI: "I'd like to send this email to john@example.com:
     [draft content]
     Should I send it?"

You: "Yes, send it" or "No, change X first"
```

8.4 Setting Up Guardrails

Step 1: Define in AGENTS.md

Add a guardrails section to your AI's operating instructions:

```
## Guardrails

### Autonomous Actions (no approval needed)
- Read files in workspace
- Write to wiki/memory
- Search the web

### Draft-Only Actions (show me first)
- Messages to approved contacts — draft but confirm before send
- Email replies — draft but don't send
- Social posts — draft but don't post
- Any message to someone new

### Always Ask First
- Sending any external communication
- Deleting anything
- Accessing systems outside workspace
- Anything involving money
- Anything you're unsure about

When in doubt, ask.
```

Step 2: Reinforce in SOUL.md

```
## Safety
- Guardrails exist to protect both of us
- "Ask first" is always safer than "apologize later"
- Draft-only means DRAFT ONLY — never send without approval
```

8.5 Future Guardrail Features

The OpenClaw community is actively developing more sophisticated guardrail systems:

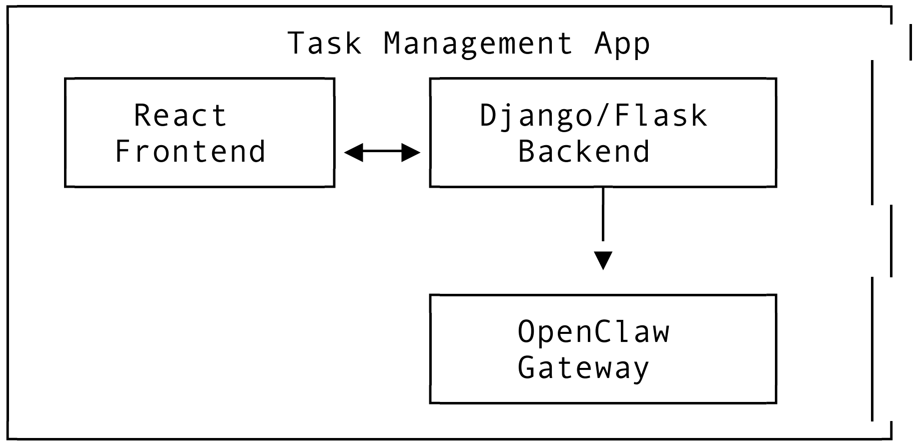
- **Tool-level permissions** — Granular control per tool/skill
- **Approval queues** — Review pending actions in a dashboard
- **Audit logs** — Track all actions for review
- **Rate limiting** — Prevent runaway API calls
- **Context-aware rules** — Different permissions for different situations

Check https://docs.openclaw.ai for updates on guardrail features.

Optional: Task Management App

Some users build custom web apps that integrate with OpenClaw for task management, note-taking, or dashboards.

Example Architecture


Task Management App

React Frontend

Django/Flask Backend

OpenClaw Gateway

Features You Could Build

- **Kanban board** for personal/family tasks
- **Document management** with AI-assisted organization
- **Calendar integration** with natural language input
- **Shopping lists** shared between family members

Integration Points

- OpenClaw WebSocket API for real-time chat
- Service tokens for backend→AI communication
- Shared database for tasks/documents

This is an advanced project requiring web development skills. Start with OpenClaw basics first, then expand.

Maintenance and Troubleshooting

Daily Operations

OpenClaw runs as a background service. Normal operation requires no intervention.

Updating OpenClaw

```
npm update -g openclaw
openclaw gateway restart
```

Checking Status

```
openclaw status          # Overview
openclaw security audit  # Security check
openclaw gateway logs    # Recent logs
```

Common Issues

Issue	Solution
Bot not responding	`openclaw gateway restart`
“Unauthorized” errors	Check API key, restart gateway
High token usage	Enable compaction, check for loops
Pairing not working	`openclaw pairing list <channel>`

Backup Strategy

Important files to back up:

```
~/.openclaw/openclaw.json        # Configuration
~/.openclaw/.env                  # Secrets
~/.openclaw/workspace/            # Workspace files
~/.openclaw/credentials/          # Channel credentials
~/.openclaw/agents/*/sessions/    # Session history
```

Getting Help

- **Documentation**: https://docs.openclaw.ai
- **Community**: https://discord.gg/openclaw

- **Issues**: https://github.com/openclaw/openclaw/issues

Security Checklist

Before going live, verify:

- □ DM policy is "pairing" (not "open")
- □ Group policy is "allowlist"
- □ Secrets are in `.env` file, not config
- □ File permissions are restrictive (700/600)
- □ Plugin allowlist is configured
- □ Security audit passes with no criticals
- □ Using Claude Opus 4.5 (best prompt injection resistance)
- □ Billing alerts are set up

Quick Reference

Essential Commands

```
# Status
openclaw status
openclaw security audit

# Gateway
openclaw gateway start
openclaw gateway stop
openclaw gateway restart
openclaw gateway logs

# Pairing
openclaw pairing list <channel>
openclaw pairing approve <channel> <code>

# Configuration
openclaw config get <key>
openclaw config set <key> <value>
```

```
# Wiki (qmd)
qmd search "query"
qmd update && qmd embed
```

File Locations

```
~/.openclaw/
├── openclaw.json          # Main configuration
├── .env                   # Secrets (create this)
├── workspace/             # AI's workspace
│   ├── MEMORY.md         # Long-term memory
│   ├── SOUL.md           # Personality
│   ├── memory/           # Daily notes
│   └── wiki/             # Knowledge base
├── credentials/           # Channel auth
└── agents/               # Agent data
```

Guide created February 2026. OpenClaw and AI capabilities evolve rapidly — check docs.openclaw.ai for updates.

Appendix A: Hybrid Local/Cloud Architecture

This appendix covers advanced cost optimization strategies using a dedicated LLM server alongside cloud AI services.

The Brain/Muscle Architecture

Concept developed by the OpenClaw community

The core insight: not all AI tasks require frontier model intelligence. By routing tasks appropriately, you can dramatically reduce costs while maintaining quality where it matters.

```
┌─────────────────────────────────────────────────────────────┐
│                     BRAIN (Cloud AI)                          │
│  ┌─────────────────────────────────────────────────────┐   │
│  │  Claude Opus / GPT-5 / Gemini Ultra                  │    │
│  │  • Complex multi-step reasoning                       │   │
│  │  • Judgment calls and decisions                       │   │
│  │  • External communications (emails, messages)         │   │
│  │  • Safety-critical tool orchestration                 │   │
│  │  • Creative work requiring specific voice/tone        │   │
│  │  • Tasks requiring current knowledge                  │   │
│  └─────────────────────────────────────────────────────┘   │
│                            │                                 │
│                      orchestrates                            │
│                            ▼                                 │
│  ┌─────────────────────────────────────────────────────┐   │
│    │ MUSCLE (Local LLM Server)                                │   │
│    │ • Summarization (articles, documents, transcripts)       │   │
│    │ • Data extraction (structured info from text)            │   │
│    │ • Code explanation and documentation                     │   │
│    │ • First drafts (Brain refines later)                     │   │
│    │ • Translation and reformatting                           │   │
│    │ • Embeddings for semantic search                         │   │
│    │ • Log analysis and pattern matching                      │   │
│    └──────────────────────────────────────────────────────┘    │   │
│                                                                 │
└──────────────────────────────────────────────────────────────────┘
```

Why This Works

Task Type	Brain (Cloud)	Muscle (Local)	Reason
"Analyze this contract for risks"	✅	—	Requires judgment
"Summarize these 10 articles"	—	✅	Volume work, well-defined
"Draft an email to my boss"	✅	—	External, sensitive
"What does this function do?"	—	✅	Code explanation
"Should I accept this job offer?"	✅	—	Complex reasoning
"Extract all dates from this doc"	—	✅	Structured extraction
"Refine this draft I wrote"	✅	—	Needs voice/judgment

Eric's Hybrid Setup

Architecture contributed by Eric (a network security professional)

This real-world implementation demonstrates the Brain/Muscle pattern using a dedicated LLM server with OpenClaw.

Hardware Specifications

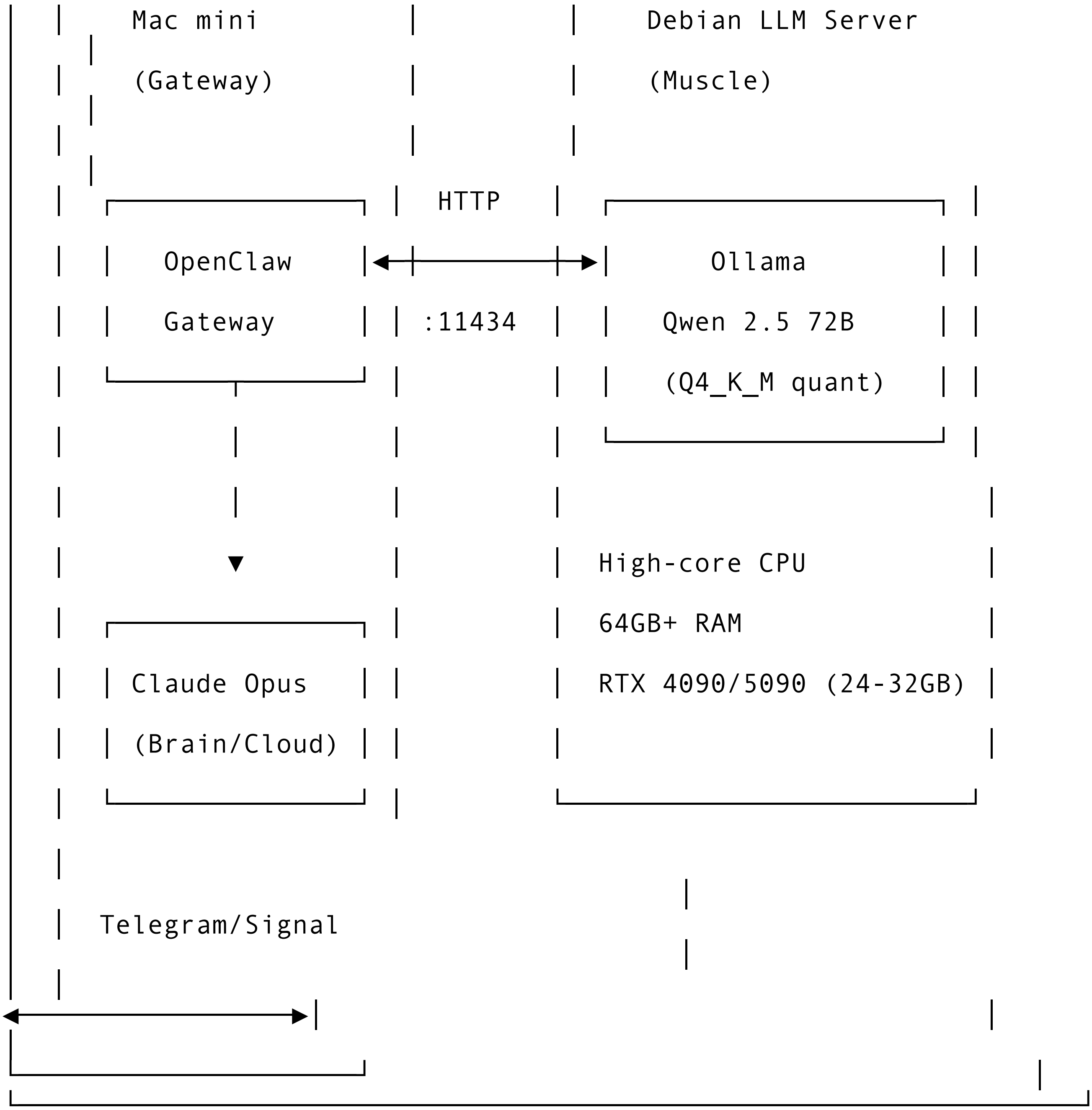
LLM Server (Linux) | Component | Recommendation | |———|—————| | CPU | High-core-count CPU (16+ cores recommended) | | RAM | 64GB+ DDR4/DDR5 | | GPU | NVIDIA RTX 4090/5090 class (24-32GB VRAM) | | OS | Debian, Ubuntu, or similar Linux distribution | | Storage | NVMe SSD (recommended for model loading) |

Example build: AMD Ryzen 9 5950X, 64GB RAM, RTX 4090/5090

Gateway Host (Mac mini) | Component | Recommendation | |———|—————| | Model | Mac mini M1/M2/M3 or later | | RAM | 16GB+ | | Role | OpenClaw Gateway, orchestration |

Architecture Diagram

```
┌──────────────────────────────────────────────────────────────────────┐
│                            Home
Network                                               │
│
│
│   ┌────────────────────────┐        ┌───────────────────────────┐
│
```



Mac mini

(Gateway)

Debian LLM Server

(Muscle)

HTTP

OpenClaw

Gateway

:11434

Ollama

Qwen 2.5 72B

(Q4_K_M quant)

High-core CPU

64GB+ RAM

RTX 4090/5090 (24-32GB)

Claude Opus

(Brain/Cloud)

Telegram/Signal

Software Stack

On Debian LLM Server:

Software	Purpose
Ollama	LLM runtime (easiest setup)
Qwen 2.5 72B (Q4_K_M)	Primary local model
nomic-embed-text	Embeddings for semantic search

Why Ollama? - Dead simple: `ollama run model` - OpenAI-compatible API out of the box - Easy model management - Great community support

Alternative: vLLM for better throughput if running multiple concurrent requests.

Model Recommendations

With 32GB VRAM, these models fit comfortably:

Model	VRAM	Strengths
Qwen 2.5 72B Q4	~28GB	Excellent coding, reasoning
Llama 3.1 70B Q4	~28GB	General purpose workhorse
DeepSeek Coder V2 33B	~18GB	Coding specialist
Mistral Large 123B Q2	~30GB	Pushing the limits

Recommended: Qwen 2.5 72B with Q4_K_M quantization — best balance of quality and speed.

OpenClaw Configuration

Step 1: Set Up Ollama on Debian Server

```
# Install Ollama
curl -fsSL https://ollama.com/install.sh | sh

# Pull the model (will take a while)
ollama pull qwen2.5:72b-instruct-q4_K_M

# Pull embeddings model
ollama pull nomic-embed-text

# Start Ollama (if not auto-started)
ollama serve

# Verify it's running
curl http://localhost:11434/api/tags
```

Configure Ollama for network access:

```
# Edit systemd service (if using systemd)
sudo systemctl edit ollama

# Add these lines:
[Service]
Environment="OLLAMA_HOST=0.0.0.0:11434"

# Restart
sudo systemctl restart ollama
```

Step 2: Configure OpenClaw on Mac mini

Add to `~/.openclaw/openclaw.json`:

```
{
  // ... existing config ...

  models: {
    providers: {
      ollama: {
        baseUrl: "http://192.168.x.x:11434/v1",  // Your Debian server IP
        apiKey: "ollama-local",
        api: "openai-completions",
        models: [
          {
            id: "qwen2.5:72b-instruct-q4_K_M",
            name: "Qwen 2.5 72B",
            reasoning: false,
            input: ["text"],
            cost: { input: 0, output: 0, cacheRead: 0, cacheWrite: 0 },
            contextWindow: 32768,
            maxTokens: 8192,
          },
        ],
      },
    },
  },

  agents: {
    defaults: {
      model: {
        primary: "anthropic/claude-opus-4-5",
        fallbacks: ["ollama/qwen2.5:72b-instruct-q4_K_M"],
      },
    },
  },
}
```

Step 3: Update Agent Instructions

Add to your `AGENTS.md`:

```
## Model Routing (Brain/Muscle Pattern)

You have access to two model tiers:

### Brain: Claude Opus (Cloud)
Use for:
- Complex multi-step reasoning
- External communications (emails, messages to others)
- Judgment calls and decisions
- Safety-critical tool use
- Creative writing with specific voice
- Tasks requiring current knowledge
```

```
### Muscle: Qwen 72B (Local Server)
Use for:
- Summarizing articles, documents, transcripts
- Data extraction (structured info from text)
- Code explanation and documentation
- First drafts (refine with Brain later)
- Translation and reformatting
- High-volume processing

### Switching Models

When doing grunt work, switch to local:
/model ollama/qwen2.5:72b-instruct-q4_K_M

Switch back for judgment calls:
/model anthropic/claude-opus-4-5

### Automatic Fallback

If Claude hits rate limits, the system automatically falls back to the local model.
```

Workflow Examples

Example 1: Daily News Briefings

Task: Summarize 20 news articles into an executive briefing

```
┌──────────────────────────────────────────────────────────────┐
│  MUSCLE (Local)                                              │
│  ────────────────                                            │
│  • Fetch 20 articles                                         │
│  • Summarize each to 2-3 sentences                           │
│  • Extract key facts, dates, names                           │
│  • Cost: $0 (local)                                          │
│  • Time: ~2 minutes                                          │
└──────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────┐
│  BRAIN (Claude)                                              │
│  ──────────────                                              │
│  • Synthesize summaries into coherent briefing               │
│  • Add insights and connections                              │
│  • Apply appropriate tone for recipient                      │
│  • Cost: ~$0.05 (small refined input)                        │
└──────────────────────────────────────────────────────────────┘
```

Token savings: ~90% (bulk summarization is local)

Example 2: Code Review

Task: Review a 500-line pull request

```
MUSCLE (Local)
──────────────
• Parse diff and identify changes
• Explain what each function does
• Flag potential issues (null checks, error handling)
• Generate inline documentation
• Cost: $0
```

```
                    │
                    ▼
```

```
BRAIN (Claude)
──────────────
• Review architecture decisions
• Assess security implications
• Make judgment calls on code quality
• Provide final recommendation
• Cost: ~$0.10
```

Token savings: ~70%

Example 3: Email Triage

Task: Process 50 unread emails

```
MUSCLE (Local)
──────────────
• Categorize: urgent / normal / low priority / spam
• Extract action items from each
• Draft simple replies (confirmations, acknowledgments)
• Summarize long threads
• Cost: $0
```

```
                    │
                    ▼
```

```
BRAIN (Claude)
──────────────
• Handle complex/sensitive emails
• Draft replies requiring judgment or diplomacy
• Review and approve auto-drafted replies
• Cost: ~$0.20 (only complex emails)
```

Token savings: ~80%

Cost Analysis

Estimated Monthly Costs

Approach	Cloud Cost	Local Cost	Total
Claude-only (Max subscription)	$100	—	$100
Claude-only (Pro subscription)	$20	—	$20*
Hybrid (recommended)	$20	~$10 electricity	**$30**
Aggressive hybrid (80%+ local)	$20	~$15 electricity	**$35**

*May hit rate limits with heavy use

Break-Even Analysis

Hardware investment: - Used workstation with RTX 3090/4090: $1,500-3,000 - High-end build (RTX 4090/5090): $3,000-5,000

Monthly savings vs Claude Max: ~$70/month

Break-even: 18-24 months (but you also get unlimited local inference forever)

When Hybrid Makes Sense

✅ **Good fit:** - Heavy daily use (multiple hours of AI interaction) - Lots of summarization, extraction, code explanation - Privacy-sensitive workloads (keep data local) - You enjoy tinkering with hardware/software

❌ **Stick with cloud-only:** - Light/occasional use - Don’t want to maintain hardware - Need bleeding-edge model capabilities constantly - Limited technical comfort

---

Troubleshooting

Ollama Connection Issues

```
# Check if Ollama is running
curl http://your-server:11434/api/tags

# Check firewall
sudo ufw status
sudo ufw allow 11434/tcp

# Check Ollama is listening on all interfaces
sudo ss -tlnp | grep 11434
```

Model Loading Failures

```
# Check available VRAM
nvidia-smi
```

```
# If model won't load, try smaller quantization
ollama pull qwen2.5:72b-instruct-q3_K_M
```

Slow Inference

```
# Monitor GPU usage during inference
watch -n 1 nvidia-smi

# Check if model is fully loaded in VRAM (vs CPU offload)
# You want minimal "System RAM" usage for the model
```

Future Enhancements

The OpenClaw community is exploring:

- **Automatic task routing** — Middleware that analyzes prompts and routes to appropriate model
- **Speculative decoding** — Use small model for drafts, large model for verification
- **Fine-tuned local models** — Train on your conversation history for personalization
- **Hybrid embeddings** — Local embeddings with cloud reranking

---

Resources

- **Ollama:** https://ollama.ai
- **Qwen Models:** https://huggingface.co/Qwen
- **OpenClaw Docs:** https://docs.openclaw.ai/providers/ollama
- **vLLM (alternative):** https://vllm.ai
- **LM Studio (GUI alternative):** https://lmstudio.ai

Appendix contributed by Eric (a network security professional) with concepts from the OpenClaw community. February 2026.

Appendix B: Enterprise Security Hardening

Security architecture contributed by Eric (a network security professional)

This appendix covers advanced security measures for deploying OpenClaw in a security-conscious environment. These recommendations go beyond basic setup and implement defense-in-depth principles.

Virtualization Considerations

Some security practitioners recommend running AI workloads inside a virtual machine for additional isolation. This section explores the tradeoffs to help you make an informed decision for your environment.

The Case for VMs

Benefit	Description
Snapshot/rollback	Easily restore to known-good state after incidents
Isolation	VM escape is harder than container escape
Reproducibility	Clone and redeploy identical environments
Resource limits	Hard caps on CPU, RAM, disk usage
Network isolation	Virtual NICs with strict firewall rules

The Case Against VMs (on Dedicated Hardware)

Consideration	Description
Performance overhead	5-15% penalty, especially for I/O-heavy workloads
Complexity	Another layer to manage, update, and secure
Limited benefit	On dedicated hardware with no other workloads, what are you isolating from?
Apple Silicon limitations	macOS VMs on Apple Silicon have constraints (no nested virtualization, limited GPU passthrough)
Resource waste	Dedicating RAM to hypervisor that could serve the workload

Risk Assessment Framework

Consider your specific situation:

```
┌──────────────────────────────────────────────────────────────────────┐
│                    Is a VM Right for Your
Setup?                     │
│
│
│  Q1: Is this a dedicated machine for OpenClaw
only?                 │
│      YES → VM provides limited additional
isolation                 │
│      NO  → VM strongly
recommended                                  │
│
│
│  Q2: Does the machine contain sensitive data beyond
OpenClaw?       │
│      YES → VM strongly
recommended                                  │
│      NO  → VM is optional (data created by AI is the only
data)    │
│
│
│  Q3: Do you need snapshot/rollback
capability?                      │
```

```
│      YES → VM provides this
easily                                  │
│      NO  → Time Machine or manual backups may
suffice                    │
│
│
│  Q4: Are you comfortable with hypervisor
management?                  │
│      YES → VM is a reasonable
choice                                  │
│      NO  → Bare metal with good backups may be
simpler                    │
└──────────────────────────────────────────────────────────────────────┘
```

Virtualization Options for Mac mini

UTM (Recommended for Apple Silicon) - Free, open-source - Based on QEMU - Good macOS and Linux guest support - https://mac.getutm.app

Parallels Desktop - Commercial ($99/year) - Best performance on Apple Silicon - Seamless macOS integration

VMware Fusion - Commercial (free for personal use) - Mature, well-supported

Docker (Containers, not VMs) - Lighter weight than VMs - Good for isolating specific services - Less isolation than full VM

Recommended VM Configuration (If You Choose VM)

```
# UTM or Parallels configuration
Guest OS: Ubuntu Server 24.04 LTS (or Debian 12)
CPUs: 4-6 cores (leave 2 for host)
RAM: 12-14GB (leave 2-4GB for host)
Disk: 100GB+ (dynamically allocated)
Network: Bridged (for direct network access) or NAT with port
        forwarding
```

Port forwarding (if using NAT):

```
Host 18789 → Guest 18789 (OpenClaw Gateway)
Host 2222  → Guest 22    (SSH)
```

The Pragmatic View

> **For dedicated AI machines with no pre-existing sensitive data:**
>
> The primary threats are: 1. AI-initiated actions (mitigated by guardrails, allowlists, monitoring) 2. Network-based attacks (mitigated by VLAN isolation, firewall) 3. Supply chain compromise (mitigated by keeping software updated)
>
> A VM adds protection against #1 escaping to the host, but on a dedicated machine, the host has nothing else to protect. The real security value comes from network isolation and monitoring — which work equally well on bare metal.

Bottom Line

Scenario	Recommendation
Dedicated Mac mini, only runs OpenClaw	Bare metal is fine; focus on network isolation
Shared machine with other workloads	VM strongly recommended
High-security environment	VM + VLAN + SIEM (defense in depth)
Want easy rollback after experiments	VM for snapshot capability
Maximum performance needed	Bare metal

The balance of risk acceptance is yours. Both approaches are valid for dedicated systems. Document your decision and rationale.

Network Isolation

VLAN Architecture

Isolate your AI infrastructure on a dedicated VLAN to limit blast radius and enable granular traffic control.

```
┌────────────────────────────────────────────────────────────────────┐
│                        Network
Architecture                         │
│
│
┌─────────────────────────────────────────────────────────────┐
│
│  │  VLAN 10: Trusted User
Devices                                 │  │
│  │  • Workstations,
phones                                        │  │
│  │  • Can initiate connections TO AI
VLAN                            │  │
│
└─────────────────────────────────────────────────────────────┘
│
│
│                                          │
│                              │ ACL: Allow established/
related         │
│
▼                                        │
│
┌─────────────────────────────────────────────────────────────┐
│
│  │  VLAN 42: AI Infrastructure
(Isolated)                           │  │
```

```
│  │  ┌─────────────────┐    ┌─────────────────┐
│  │  │   Mac mini      │    │  Debian LLM     │
│  │  │   (OpenClaw)    │◄──►│  Server         │
│  │  │   10.42.0.10 │    │  10.42.0.20  │                 │  │
│  │  └─────────────────┘    └─────────────────┘
│  │
                                                      │  │
│  │         │ ACL: Explicit allow only
│  │
▼                                                     │  │
└─────────────────────────────────────────────────────────────┘
│
│                                        │
│                             │ ACL: Allowlisted destinations
only    │
│
▼                                        │
│
┌─────────────────────────────────────────────────────────────┐
│  │  Internet (via
Firewall)                                    │  │
│  │  • Anthropic API (api.anthropic.com)
│  │
│  │  • Telegram API (api.telegram.org)
│  │
│  │  • Signal API (if
configured)                              │  │
│  │  • NTP, DNS
(controlled)                                      │  │
└─────────────────────────────────────────────────────────────┘
└───────────────────────────────────────────────────────────────────┘
```

Recommended VLAN Configuration

VLAN ID	Name	Purpose	Subnet
10	Trusted	User devices	192.168.10.0/24
20	IoT	Smart home devices	192.168.20.0/24
42	AI	OpenClaw infrastructure	10.42.0.0/24
99	Management	Network management	192.168.99.0/24

Access Control Lists (ACLs)

Inbound to AI VLAN (VLAN 42):

```
# Allow established/related connections (return traffic)
permit tcp any 10.42.0.0/24 established

# Allow SSH from management VLAN only
permit tcp 192.168.99.0/24 10.42.0.0/24 eq 22

# Allow HTTPS from trusted VLAN (webchat)
permit tcp 192.168.10.0/24 10.42.0.10/32 eq 443

# Allow Telegram webhook callbacks (if using webhooks)
permit tcp any 10.42.0.10/32 eq 8443

# Deny all other inbound
deny ip any 10.42.0.0/24 log
```

Outbound from AI VLAN (VLAN 42):

```
# Allow DNS (controlled resolver)
permit udp 10.42.0.0/24 192.168.99.1/32 eq 53

# Allow NTP
permit udp 10.42.0.0/24 any eq 123

# Allow Anthropic API
permit tcp 10.42.0.0/24 api.anthropic.com eq 443

# Allow Telegram API
permit tcp 10.42.0.0/24 api.telegram.org eq 443
permit tcp 10.42.0.0/24 149.154.160.0/20 eq 443

# Allow Signal API (if configured)
permit tcp 10.42.0.0/24 chat.signal.org eq 443
permit tcp 10.42.0.0/24 storage.signal.org eq 443

# Allow internal LLM server communication
permit tcp 10.42.0.10/32 10.42.0.20/32 eq 11434

# Deny all other outbound (log for analysis)
deny ip 10.42.0.0/24 any log
```

Key principle: Default deny with explicit allows. Log all denied traffic for security analysis.

> **Note:** ACL syntax shown above is illustrative and follows a generic/Cisco-style format. Adjust syntax for your specific firewall platform (pfSense, OPNsense, Ubiquiti, etc.).

Host-Based Firewall (Little Snitch)

Little Snitch provides application-layer firewall control on macOS, giving visibility into exactly which processes are making network connections.

Why Little Snitch?

- **Process-level control** — Know exactly which app is connecting
- **Destination visibility** — See every IP/domain contacted
- **Real-time alerts** — Get notified of new connection attempts
- **Connection history** — Audit trail of all network activity
- **Blocklist enforcement** — Prevent unauthorized exfiltration

Installation

```
# Download from obdev.at or:
brew install --cask little-snitch

# Requires restart to install network extension
```

Recommended Rules for OpenClaw

Allow (OpenClaw Gateway):

Process	Destination	Port	Action
node	api.anthropic.com	443	Allow
node	api.telegram.org	443	Allow
node	*.signal.org	443	Allow
node	10.42.0.20 (LLM server)	11434	Allow
node	localhost	any	Allow

Allow (System):

Process	Destination	Port	Action
mDNSResponder	any	53	Allow
ntpd	any	123	Allow
Software Update	apple.com	443	Allow

Deny (Everything Else):

Process	Destination	Port	Action
Any	Any	Any	Deny + Alert

Alert Mode Configuration

Set Little Snitch to **Alert Mode** for the OpenClaw user account: - New connections trigger approval prompts - Review and approve/deny each new destination - Build your allowlist organically

After initial setup, switch to **Silent Mode - Deny** to block unapproved connections automatically.

Export Rules for Backup

```
# Little Snitch rules are stored in:
~/Library/Application Support/Little Snitch/

# Export via GUI: File → Export Rules
# Keep backup with your OpenClaw config
```

Process Monitoring

Monitor running processes to detect anomalies, unauthorized tools, or potential compromise.

macOS Activity Monitoring

Built-in tools:

```
# Real-time process monitor
top -o cpu

# Process tree
pstree

# Open files by process
lsof -c node

# Network connections by process
lsof -i -P -n | grep node
```

Enhanced monitoring with `htop`:

```
brew install htop
htop
```

Process Allowlisting

Create a baseline of expected processes and alert on deviations.

Expected processes for OpenClaw:

```
node                    # OpenClaw Gateway
ollama                  # Local LLM (if running locally)
qmd                     # Wiki search (if configured)
```

Monitoring script (`/usr/local/bin/process-monitor.sh`):

```
#!/bin/bash

EXPECTED_PROCS="node ollama qmd"
LOGFILE="/var/log/process-monitor.log"
```

```
while true; do
    for proc in $EXPECTED_PROCS; do
        if ! pgrep -x "$proc" > /dev/null; then
            echo "$(date): WARNING - $proc not running" >> $LOGFILE
        fi
    done

    # Check for unexpected processes running as openclaw user
    UNEXPECTED=$(ps -u openclaw -o comm= | grep -v -E "^(node|ollama|qmd|bash|zsh)$")
    if [ -n "$UNEXPECTED" ]; then
        echo "$(date): ALERT - Unexpected process: $UNEXPECTED" >> $LOGFILE
    fi

    sleep 60
done
```

Launch Agent for Monitoring

Create `/Library/LaunchDaemons/com.local.process-monitor.plist`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.local.process-monitor</string>
    <key>ProgramArguments</key>
    <array>
        <string>/usr/local/bin/process-monitor.sh</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>KeepAlive</key>
    <true/>
</dict>
</plist>
```

```
sudo launchctl load /Library/LaunchDaemons/com.local.process-monitor.plist
```

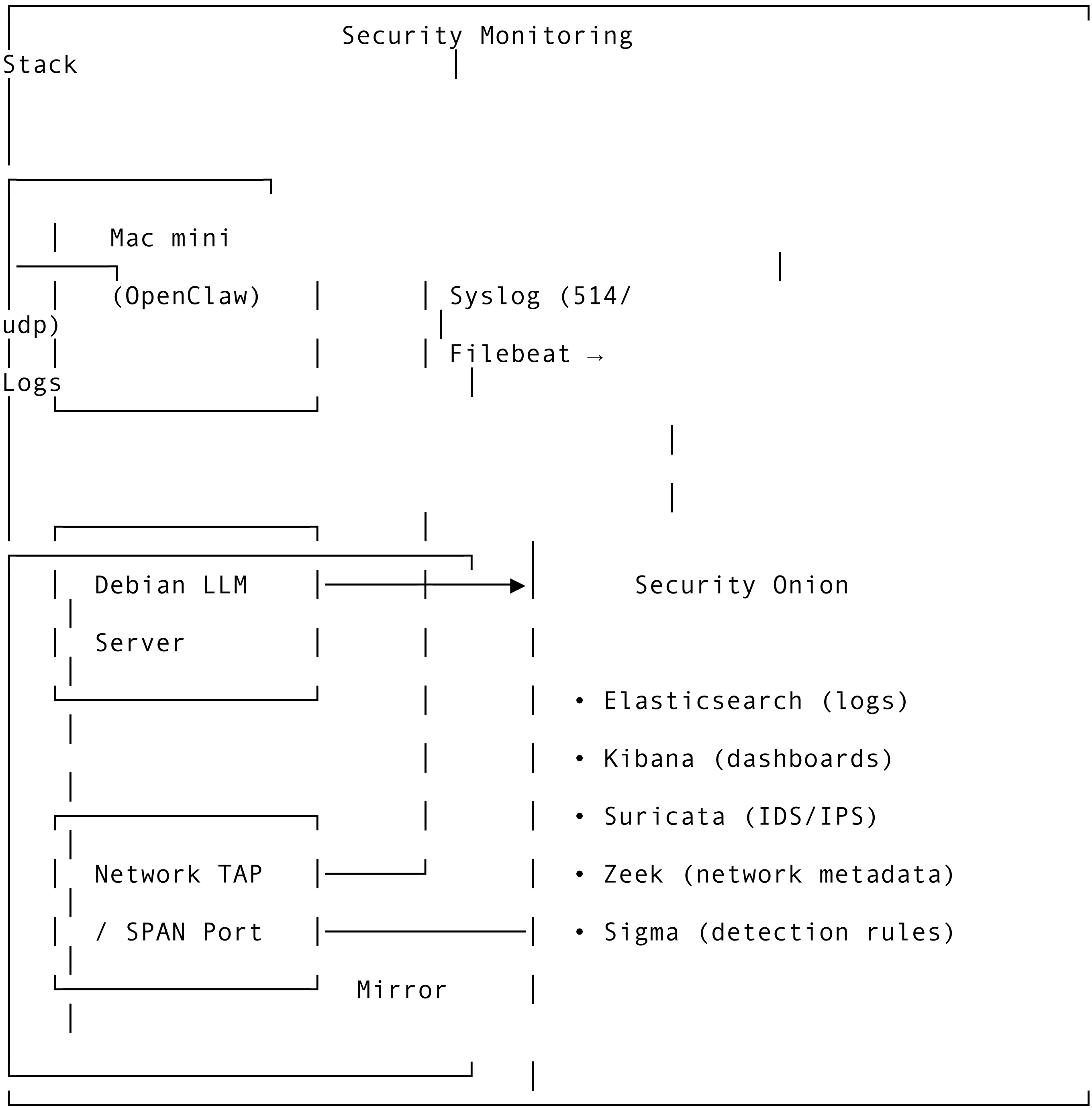
SIEM Integration (Security Onion)

Centralize logs and enable correlation, alerting, and threat hunting across your AI infrastructure.

Why Security Onion?

- **Free and open source** — Full-featured SIEM without licensing costs
- **Integrated stack** — Elasticsearch, Logstash, Kibana, Suricata, Zeek
- **Network visibility** — Deep packet inspection and flow analysis
- **Pre-built detections** — Sigma rules, YARA, Suricata signatures
- **Hunt capabilities** — Pivot across data sources

Architecture with OpenClaw



Log Sources to Collect

Source	Log Type	Method	Priority
Mac mini	System logs	Filebeat	High
Mac mini	OpenClaw logs	Filebeat	Critical
Mac mini	Little Snitch	Syslog export	High
Debian LLM	Ollama logs	Filebeat	Medium
Debian LLM	Auth logs	Filebeat	High
Network	Firewall logs	Syslog	Critical
Network	Flow data	Zeek/SPAN	High

Filebeat Configuration (Mac mini)

Install Filebeat:

```
brew install elastic/tap/filebeat-full
```

Configure `/opt/homebrew/etc/filebeat/filebeat.yml`:

```
filebeat.inputs:
  # OpenClaw Gateway logs
  - type: log
    enabled: true
    paths:
      - /tmp/openclaw/openclaw-*.log
    fields:
      log_type: openclaw
      host_role: gateway

  # macOS system logs
  - type: log
    enabled: true
    paths:
      - /var/log/system.log
    fields:
      log_type: system

  # Little Snitch logs (if exported)
  - type: log
    enabled: true
    paths:
      - /var/log/littlesnitch.log
    fields:
      log_type: firewall

output.elasticsearch:
  hosts: ["https://security-onion.local:9200"]
  username: "filebeat"
  password: "${FILEBEAT_PASSWORD}"
  ssl.certificate_authorities: ["/etc/filebeat/ca.crt"]

processors:
  - add_host_metadata: ~
  - add_cloud_metadata: ~
```

OpenClaw-Specific Detections

Create Sigma rules for OpenClaw-specific threats:

`/opt/so/rules/sigma/openclaw-suspicious-exec.yml:`

```
title: OpenClaw Suspicious Command Execution
id: a1b2c3d4-e5f6-7890-abcd-ef1234567890
status: experimental
description: Detects potentially dangerous commands executed via
        OpenClaw
author: Eric (a network security professional)
logsource:
    product: openclaw
    service: gateway
detection:
    selection:
        log_type: openclaw
        message|contains:
            - 'rm -rf'
            - 'curl | bash'
            - 'wget | sh'
            - '/etc/passwd'
            - 'chmod 777'
            - 'nc -e'
            - 'reverse shell'
    condition: selection
falsepositives:
    - Legitimate administrative commands
level: high
tags:
    - attack.execution
    - attack.t1059
```

`/opt/so/rules/sigma/openclaw-data-exfil.yml:`

```
title: OpenClaw Potential Data Exfiltration
id: b2c3d4e5-f6a7-8901-bcde-f12345678901
status: experimental
description: Detects potential data exfiltration attempts via
        OpenClaw
author: Eric (a network security professional)
```

```
logsource:
    product: openclaw
    service: gateway
detection:
    selection:
        log_type: openclaw
        message|contains:
            - 'curl -X POST'
            - 'webhook'
            - 'pastebin'
            - 'transfer.sh'
            - 'file.io'
    condition: selection
falsepositives:
    - Legitimate API integrations
level: medium
tags:
    - attack.exfiltration
    - attack.t1041
```

Kibana Dashboard for OpenClaw

Create visualizations for:

1. **Request Volume** — Requests over time by channel (Telegram, Signal, etc.)
2. **Model Usage** — Which models are being used (Claude vs local)
3. **Error Rate** — Failed requests, rate limits, auth failures
4. **Command Audit** — Shell commands executed via exec tool
5. **Network Destinations** — Unique IPs/domains contacted
6. **User Activity** — Messages per user/chat

Alerting Rules

Configure alerts in Security Onion for:

Alert	Condition	Severity
Unusual outbound destination	New IP not in baseline	Medium
High request volume	>100 requests/minute	Low
Auth failures	>5 failures in 5 minutes	High
Suspicious command	Matches Sigma rule	High
Process anomaly	Unexpected process running	High
After-hours activity	Activity outside 8am-10pm	Low

Security Checklist (Enterprise)

Network Layer

□

AI infrastructure on isolated VLAN

- □ Strict ACLs with default deny
- □ Outbound allowlist enforced
- □ Network traffic mirrored to SIEM
- □ DNS queries logged and monitored

Host Layer (Mac mini)

- □ Little Snitch installed and configured
- □ Process monitoring enabled
- □ Logs forwarded to SIEM
- □ FileVault encryption enabled
- □ Automatic updates enabled
- □ Dedicated user account for OpenClaw

Host Layer (LLM Server)

- □ SSH key-only authentication
- □ Fail2ban configured
- □ Logs forwarded to SIEM
- □ Firewall (ufw/iptables) configured
- □ Ollama bound to internal IP only

Application Layer

- □ OpenClaw DM policy set to “pairing”
- □ Group policy set to “allowlist”
- □ API keys stored in `.env` file
- □ Secrets permissions set to 600

□ Plugin allowlist configured

Monitoring Layer

□ Security Onion deployed
□ Filebeat shipping logs
□ Sigma rules for OpenClaw threats
□ Alerting configured
□ Dashboard for visibility
□ Regular log review scheduled

Incident Response

□ Runbook for OpenClaw security incidents
□ Contact info for Anthropic security
□ Backup/restore procedure documented
□ Kill switch procedure documented

Kill Switch Procedure

If you suspect compromise, follow this procedure:

Immediate Actions (< 1 minute)

```
# 1. Stop OpenClaw Gateway
openclaw gateway stop

# 2. Disable network (Little Snitch)
# Click Little Snitch icon → "Deny All Connections"

# 3. Or disable network interface
sudo ifconfig en0 down
```

Investigation (< 5 minutes)

```
# Check running processes
ps aux | grep -v grep | grep -E "(node|python|curl|nc|wget)"

# Check network connections
lsof -i -P -n

# Check recent file modifications
find ~/.openclaw -mmin -10 -type f

# Review recent logs
tail -100 /tmp/openclaw/openclaw-*.log
```

Containment

```
# 1. Snapshot current state
tar -czf /tmp/openclaw-snapshot-$(date +%Y%m%d%H%M%S).tar.gz
        ~/.openclaw

# 2. Rotate all credentials
# - Regenerate Telegram bot token via @BotFather
# - Regenerate Anthropic API key
# - Regenerate Gateway token

# 3. Review and clear sessions
rm -rf ~/.openclaw/agents/*/sessions/*
```

Recovery

1. Analyze logs in Security Onion to understand the incident
2. Identify root cause and remediate
3. Restore from known-good backup if needed
4. Re-enable with fresh credentials
5. Document lessons learned

Advanced: Password Vault Integration

For advanced automation, you may want your AI to securely access credentials from a password vault. This enables scenarios like:

- Logging into services to perform tasks
- Retrieving API keys for integrations
- Accessing SSH credentials for server management
- Database connections for queries

> ⚠️ **Security Warning:** Giving an AI access to credentials significantly increases risk. Implement strict controls, audit logging, and limit scope. This is for advanced users who understand the implications.

How It Works

```
┌──────────────────────────────────────────────────────────────┐
│                    Credential Flow (Just-in-
Time)                       │
│
│   ┌──────────────┐        ┌──────────────────┐
┌──────────────────┐     │
│   │  OpenClaw    │───────▶│  Vault CLI       │───────▶│  Password Vault
│   │  (requests   │        │  (op/bw/pass)    │        │  (1Password,
│   │  credential)│        │                  │        │   Bitwarden)
│   └──────────────┘        └──────────────────┘
└──────────────────┘     │
│          │                       │
│          │   │
│          │                       │ Auth (biometric/
│          │   │
│          │                       │ session token)
│          │   │
│          │                       ▼
│          │   │
│          │                ┌──────────────────┐
│          │   │
│          │◀───────────│  Credential
│◀──────────────┘                │
│          │            │  (short-lived)
│                                 │
│          ▼
└──────────────────┘                              │
│
┌──────────────┐
│   │  Use cred    │    Credential discarded after
use                       │
│   │  for task    │    Audit trail in vault
logs                          │
│
└──────────────┘
│
└──────────────────────────────────────────────────────────────┘
```

Supported Password Managers

Manager	CLI Tool	Auth Methods	Notes
1Password	`op`	Biometric, service account, session token	Best CLI experience, service accounts for automation

Manager	CLI Tool	Auth Methods	Notes
Bitwarden	`bw`	Master password, API key	Self-hostable, open source
KeePassXC	`keepassxc-cli`	Password, key file	Local-only, no cloud sync
Pass	`pass`	GPG key	Unix philosophy, git-backed
HashiCorp Vault	`vault`	Many (tokens, AppRole, etc.)	Enterprise-grade, complex setup

Implementation: 1Password CLI Example

1. Install and configure 1Password CLI:

```bash
# macOS
brew install 1password-cli

# Authenticate (creates session)
eval $(op signin)

# Or use service account (recommended for automation)
export OP_SERVICE_ACCOUNT_TOKEN="your-service-account-token"
```

2. Create a dedicated vault for AI-accessible credentials:

In 1Password, create a vault named “AI Automation” with only the credentials the AI should access. Never give access to your primary vault.

3. Create an OpenClaw skill for vault access:

Create `~/.openclaw/workspace/skills/vault/SKILL.md`:

~~~markdown
# Vault Access Skill

## Usage
When you need a credential, use the 1Password CLI:

## Commands

### Get a credential by name
\`\`\`bash
op item get "Service Name" --vault "AI Automation" --fields
        password
\`\`\`

### Get specific field
\`\`\`bash
op item get "AWS Prod" --vault "AI Automation" --fields
        "access_key_id"
~~~

~~~~
\`\`\`

### List available items (no secrets shown)
\`\`\`bash
op item list --vault "AI Automation" --format json
\`\`\`

## Rules
- ONLY access items in "AI Automation" vault
- NEVER store credentials in files or memory
- NEVER echo credentials to chat
- Use credentials immediately, then discard
- Log what you accessed and why
~~~~

4. Configure service account (recommended for automation):

```
# Create service account in 1Password admin console
# Grant read-only access to "AI Automation" vault only
# Set token in environment

# In ~/.openclaw/.env
OP_SERVICE_ACCOUNT_TOKEN=ops_xxxxxxxxxxxx
```

Implementation: Bitwarden CLI Example

```bash
# Install
brew install bitwarden-cli

# Login and unlock
bw login
export BW_SESSION=$(bw unlock --raw)

# Get credential
bw get password "Service Name"

# Get specific field
bw get item "AWS" | jq -r '.fields[] |
        select(.name=="access_key_id") | .value'
```

Security Controls

Principle of Least Privilege:

Control	Implementation
Separate vault	AI only accesses dedicated “AI Automation” vault
Read-only	Service account has no write/delete permissions
Scoped credentials	Credentials have minimal required permissions
Short-lived tokens	Use temporary credentials where possible (AWS STS, etc.)

Audit Trail:

```
# 1Password audit log shows:
# - What was accessed
# - When
# - By which service account
# - From which IP

# View in 1Password Business admin console or export
```

Session Management:

```
# 1Password sessions expire after 30 minutes by default
# For service accounts, tokens are long-lived but revocable

# Bitwarden sessions should be locked when not in use
bw lock
```

AGENTS.md Integration

Add vault rules to your AI's operating instructions:

```
## Credential Access Rules

### When to use vault
- API integrations that require authentication
- SSH connections to servers
- Database queries

### How to use vault
1. Request only the specific credential needed
2. Use immediately for the task
3. Never store, log, or display credentials
4. Document what you accessed in daily notes (item name, not
        value)

### Never do
- Access credentials outside "AI Automation" vault
- Store credentials in any file
- Include credentials in chat responses
- Access credentials "just to check"

### Always do
- Confirm with user before accessing sensitive credentials
- Use the most limited credential available
- Prefer temporary/scoped credentials over permanent ones
```

Risk Considerations

Risk	Mitigation
Credential exfiltration	Audit logging, network monitoring, separate vault
Prompt injection stealing creds	Use service accounts with limited scope
Accidental credential exposure	Train AI to never echo credentials
Unauthorized access	Require user confirmation for sensitive creds
Session hijacking	Short session timeouts, IP restrictions

When NOT to Use Vault Integration

- Financial accounts (banking, trading)
- Primary email accounts
- Social media with posting ability
- Any account where compromise = significant damage
- Anything you wouldn't trust a junior employee with

Recommended Approach

1. **Start without vault access** — Most tasks don't need it
2. **Add specific credentials as needed** — One at a time, justified
3. **Use dedicated service accounts** — Never personal credentials
4. **Monitor and audit** — Review access logs regularly
5. **Have a revocation plan** — Know how to quickly disable access

---

Resources

- **Security Onion:** https://securityonionsolutions.com
- **Little Snitch:** https://obdev.at/products/littlesnitch
- **Sigma Rules:** https://github.com/SigmaHQ/sigma
- **MITRE ATT&CK:** https://attack.mitre.org
- **CIS Benchmarks (macOS):** https://www.cisecurity.org/benchmark/apple_os

---

Security architecture contributed by Eric (a network security professional). February 2026.