Ai Launch Desk

LEARN • GROW • AUTOMATE

SUBSCRIBER-EXCLUSIVE LAB GUIDE

Automation Lab Guide

Setting Up Your Local Demo Environment with OpenClaw + Claude API

EDITION Advanced Practitioner

VERSION 1.1

ISSUED June 2026

ISSUED FOR Usman Zaheer

DEMO ENVIRONMENT ONLY

For learning and prototyping. Build production deployments on a dedicated VPS with your own credentials and a hardened security posture.

Table of Contents

1. Introduction & What You'll Build

This lab gets a working AI-agent runtime — OpenClaw — wired to Anthropic's Claude API on your local machine. By the end you'll have a sandbox that fetches, summarizes, classifies, and chains tasks, all driven by structured JSON and Claude Haiku for near-zero cost.

The build is intentionally **local-only**. No domain, no cloud server, no exposed ports. Everything runs on `localhost` and the only external service touched is `api.anthropic.com` over HTTPS. This is the right scope for understanding the architecture before committing it to production hardware. Section 8 covers the bridge to a VPS.

1.1 What you'll have at the end

- A locally running OpenClaw CLI + Gateway daemon.
- A valid Anthropic API key wired into OpenClaw via environment variable.
- A working web-article summarizer task.
- A working batch sentiment classifier task.
- (Optional) n8n connected to OpenClaw for visual workflow composition.
- A cost-capped, monitored configuration that won't surprise you on the next invoice.

1.2 Stack overview

Tool	Role	Required?
OpenClaw	Agent runtime — orchestrates tasks, hosts the local Gateway, calls Claude	Yes
Claude API (Anthropic)	Reasoning engine — summarization, classification, planning, generation	Yes
n8n	Visual workflow composer for chaining HTTP-driven steps	Optional
VS Code (or any editor)	Config + task JSON editing	Recommended
Git + a private repo	Versioning your task JSON and configs (with .gitignore for secrets)	Recommended

> **REMINDER — DEMO SCOPE**
>
> Everything is localhost-bound. No port is internet-reachable. Experiment freely within these boundaries.

2. Prerequisites

2.1 System requirements

- **OS.** macOS 12+ | Ubuntu 20.04+ / Debian | Windows 10/11 (WSL2 strongly preferred over native).
- **Node.js.** v22.19+ or v24+. The installer will fetch v22 LTS if it's missing.
- **RAM.** 4 GB minimum; 8 GB if running n8n alongside.
- **Disk.** ~500 MB free.
- **Network.** Outbound HTTPS to api.anthropic.com and openclaw.ai.
- **Terminal.** Bash/Zsh on Unix; PowerShell 5.1+ or WSL2 bash on Windows.

> **PRO TIP — WINDOWS USERS**
>
> WSL2 sidesteps every Windows-specific permission and path issue you'd otherwise hit. From an Administrator PowerShell: wsl --install, then reboot. Use WSL2's Ubuntu for the rest of this guide.

2.2 Accounts

- **Anthropic Console.** Free signup at console.anthropic.com. Pay-per-token billing; no subscription.
- **Email.** Required by the OpenClaw onboarding wizard.
- **GitHub.** Optional — only if you plan to build OpenClaw from source.

2.3 Cost awareness — read before generating any key

Claude API is per-token, both input and output. A full pass through this guide's sample workflows on Claude Haiku 4.5 should cost under **$0.05**.

Before generating an API key:

- Default to claude-haiku-4-5 for everything in this lab. It's ~95% cheaper than Sonnet and ~99% cheaper than Opus while being plenty capable for these tasks.
- Set a $5 monthly spend cap in the Anthropic Console (Section 6.1) before you create your first key. Spend caps must exist before the key is live, not after.
- Avoid Sonnet/Opus for demo workflows. Reserve them for production tasks where reasoning quality justifies the 4–18× cost.

> **FREE TIER NOTE**
>
> Anthropic includes a limited free tier for new accounts — useful for the very first test calls but not for sustained experimentation. To run the full lab repeatedly, add a payment method and set the spend cap in the same session.

---

3. Installing OpenClaw Locally

OpenClaw's installer detects your OS, ensures Node.js is present, drops the CLI + Gateway daemon into your path, and runs an interactive onboarding wizard. End-to-end install on a modern machine is ~90 seconds.

3.1 Quick install

macOS / Linux / WSL2:

```
curl -fsSL https://openclaw.ai/install.sh | bash
```

Windows (PowerShell as Administrator):

```
iwr -useb https://openclaw.ai/install.ps1 | iex
```

The installer:

1. Detects OS + architecture (Intel, Apple Silicon, ARM, x86_64).
2. Installs Node.js v22 LTS if not present.
3. Installs the OpenClaw CLI globally.
4. Registers the OpenClaw Gateway daemon (systemd / launchd / Windows service).
5. Launches the interactive onboarding wizard.

> **PRO TIP — INSPECT BEFORE PIPING TO SHELL**
>
> Open https://openclaw.ai/install.sh (or install.ps1) in a browser first. This is good hygiene for any curl | bash style installer regardless of vendor.

3.2 Verifying the installation

Open a **new** terminal (so PATH is refreshed) and run all three:

```
openclaw --version       # → openclaw v2.4.x
openclaw doctor          # → green checks on Node.js, Gateway, key, policies
openclaw gateway status  # → RUNNING, listening on :3000
```

Expected gateway status output:

```
Gateway Status:    RUNNING
Listening on:      http://localhost:3000
Uptime:            00:02:14
Active connections: 0
```

SUCCESS

Three green commands = healthy install. Proceed to Section 4.

3.3 Alternative install methods

Method	Command	When to use
npm	npm install -g @openclaw/cli	Node.js 22+ already installed; standard global install
pnpm	pnpm add -g @openclaw/cli	pnpm is your default package manager
bun	bun add -g @openclaw/cli	Faster install on macOS/Linux with bun installed
From source	git clone github.com/openclaw/openclaw npm install && npm run build && npm link	You want to read or modify the source
Docker	docker pull openclaw/openclaw:latest docker run -p 3000:3000 openclaw/openclaw:latest	Sandboxed/CI environment, clean teardown
Local prefix	install-cli.sh → installs under ~/.openclaw only	Shared machines, full isolation from system Node.js

ADVANCED — DOCKER IS THE RIGHT CHOICE FOR PRODUCTION REPRODUCIBILITY

Even for this local demo, running OpenClaw in Docker gets you immutable infrastructure semantics: known-good image hash, clean teardown, easy version pinning. The trade-off is one extra layer of port-mapping and volume mounts. If you're comfortable with Docker, prefer it.

3.4 Troubleshooting

Issue	Likely cause	Fix
Permission denied on macOS/Linux	Script lacks write to /usr/local	sudo-prefix, or install Node.js via nvm first to avoid root

Issue	Likely cause	Fix
Node version error	Installed Node.js < v22	nvm install 22 && nvm use 22, re-run installer
PowerShell execution policy error	Policy set to Restricted	Set-ExecutionPolicy RemoteSigned -Scope CurrentUser, re-run
Gateway not starting	Port 3000 in use, or daemon not registered	openclaw gateway install then openclaw gateway start. Rebind: edit ~/.openclaw/openclaw.json gateway.port
API key not recognized	Env var unset or in a different shell	export ANTHROPIC_API_KEY=sk-ant-... — add to ~/.bashrc or ~/.zshrc to persist
command not found: openclaw after install	PATH not refreshed in current shell	Open a new terminal, or source ~/.bashrc / source ~/.zshrc

4. Connecting OpenClaw to Claude via API Token

4.1 Generate the Anthropic API key

6. Sign in at console.anthropic.com.
7. Sidebar → API Keys.
8. Create Key. Name it descriptively (e.g., openclaw-demo-key).
9. Copy immediately. The key starts with sk-ant- and is shown exactly once. Store in 1Password, Bitwarden, or your secrets manager of choice.
10. If you'll go beyond free tier, go to Billing → add payment method → return to Section 6.1 to set the spend cap before the key sees production traffic.

> **WARNING — KEY HANDLING**
>
> The API key has full billing authority on your Anthropic account. Never paste it in chat, Slack, email, screenshots, or commits. If it leaks, rotate immediately: Console → API Keys → Delete → Create New.

4.2 Configure OpenClaw

Option A — Interactive wizard (cleanest):

```
openclaw onboard
```

Wizard steps:

11. Provider → Anthropic (Claude).
12. Paste key → wizard validates against the live API.
13. Model → claude-haiku-4-5.
14. Chat platform → skip, or pick WebChat for a localhost:3000/chat browser UI.

Option B — Manual config file:

Path:

- **macOS / Linux:** ~/.openclaw/openclaw.json
- **Windows:** %APPDATA%\openclaw\openclaw.json

Minimal config:

```
{
 "provider": "anthropic",
 "model": "claude-haiku-4-5",
 "apiKey": "${ANTHROPIC_API_KEY}",
 "maxTokensPerRequest": 2000,
 "gateway": {
  "port": 3000,
  "enabled": true
 }
}
```

PRO TIP — ALWAYS USE ${ANTHROPIC_API_KEY}, NEVER THE RAW KEY

The ${...} syntax tells OpenClaw to interpolate from your shell environment at runtime. This way the config file is safe to commit, share, or copy between machines.

Add to your shell profile:

- bash/zsh: export ANTHROPIC_API_KEY=sk-ant-... in ~/.bashrc or ~/.zshrc
- PowerShell: $env:ANTHROPIC_API_KEY = "sk-ant-..." in your $PROFILE

ADVANCED — SECRETS MANAGEMENT FOR PRODUCTION

Environment variables are fine for local demo. For production, push secrets into a real manager: macOS Keychain (security add-generic-password), HashiCorp Vault, AWS Secrets Manager, Doppler, or 1Password CLI (op read). OpenClaw config supports shell command substitution — wrap a 1Password or vault read invocation into the env var assignment in your service unit file.

4.3 Verify the connection

```
openclaw chat "Hello, are you connected to Claude?"
```

Expected output:

```
> Claude via Anthropic API — Connected ✓
> Model: claude-haiku-4-5
> Response: "Hello! Yes, I'm connected and ready to help with your
>           automation tasks. What would you like to build today?"
```

> **SUCCESS**
>
> Connected status + Claude response = green light for Section 5.

If you see an error:

- **401 Unauthorized** — key not loaded into the shell. Re-export, restart terminal, retry.
- **Connection refused** — Gateway not running. openclaw gateway start.
- **Model not found** — config typo. Must be exactly claude-haiku-4-5.

---

5. Running Real Automation Workflows

5.1 The task + agent execution loop

Every OpenClaw workflow follows the same five-step loop. Internalize this — it's the foundation under every agentic system you'll ever build:

15. You define a task in a JSON file: what to do, what input, what output shape, which model + parameters.
16. OpenClaw reads the task and constructs the structured prompt.
17. Claude reasons through it via Anthropic's API and returns a structured response.
18. OpenClaw parses the response and runs any downstream actions defined in the task (HTTP, file write, secondary task chain).
19. Output streams to stdout and is persisted as a timestamped JSON file under ./openclaw-output/.

> **ADVANCED — MENTAL MODEL**
>
> This is the same pattern as LangChain/LangGraph, AutoGen, CrewAI, and every other agent framework — a structured task spec, a model call, a parsed response, an effect. The frameworks differ in their abstractions over steps 4–5 (planning, memory, tool use, multi-agent handoff). Understanding this primitive deeply means you can reimplement any of them when you outgrow their abstractions.

5.2 Workflow 1 — Web article summarizer

demo-task.json:

```
{
 "task": "Summarize the following article for me",
 "input": {
  "url": "https://example.com/article",
  "format": "bullet_points",
  "max_bullets": 5
 },
 "agent_config": {
  "model": "claude-haiku-4-5",
  "max_tokens": 1000,
  "temperature": 0.3
 }
}
```

Replace the URL. Keep `temperature: 0.3` — low temperature produces focused, factual output for summarization. Raise to ~0.7 only when you want creative variation.

Run:

```
openclaw run demo-task.json
```

Review the persisted output:

```
cat ./openclaw-output/latest.json
```

> **NOTE — URL FETCHING**
>
> OpenClaw fetches the article HTML, extracts the readable text, and sends the text (not the URL) as the prompt input. Paywalled, login-gated, or CAPTCHA'd URLs will fail at the fetch step. For those, paste the article text directly into the task JSON under input.text.

5.3 Workflow 2 — Batch sentiment classifier

demo-classify.json:

```
{
 "task": "Classify each piece of customer feedback as Positive, Neutral, or Negative.
        Return results as a JSON array with text and sentiment fields.",
 "inputs": [
  "The product arrived two days early and the packaging was fantastic!",
  "It works fine but I expected more features for the price.",
  "This is the worst purchase I've made this year. Completely broken on arrival."
 ],
 "agent_config": {
  "model": "claude-haiku-4-5",
  "max_tokens": 500,
```

```
  "temperature": 0.1
 }
}
```

Run:

```
openclaw run demo-classify.json
```

Expected response:

```
[
 { "text": "The product arrived two days early...", "sentiment": "Positive" },
 { "text": "It works fine but I expected more...",  "sentiment": "Neutral"  },
 { "text": "This is the worst purchase...",         "sentiment": "Negative" }
]
```

ADVANCED — SCALING THIS PATTERN

Haiku handles batch classification at ~50 items/second at Tier 1 limits. For thousands of items, batch in groups of ~50 per request rather than one-per-request: lower API overhead, higher throughput, fewer tokens spent on the system prompt. For tens of thousands, use Anthropic's Batch API — 50% discount in exchange for asynchronous completion within 24 hours.

5.4 Connecting n8n (optional, visual workflows)

n8n is a self-hostable workflow tool that talks to OpenClaw's Gateway over HTTP. Useful when you want to chain Claude calls with non-AI steps (Slack notifications, Google Sheets writes, webhook fan-out, schedule triggers).

Local install:

```
npm install -g n8n
n8n start
```

n8n runs on http://localhost:5678. Add an HTTP Request node with:

- **URL.** http://localhost:3000/task
- **Method.** POST
- **Body type.** JSON
- **Body.** Your task JSON (e.g., the demo-task.json content above)

NOTE — LOCAL-ONLY

n8n on :5678 is also localhost-bound here. For a persistent 24/7 setup with HTTPS, see Section 8.

6. Capping API Usage — Cost Protection

Two layers of protection. Both belong before any workflow runs.

6.1 Hard spend limit in Anthropic Console

20. console.anthropic.com → Settings → Billing & Usage.
21. Spend Limits → Set Limit.
22. Enter monthly cap (recommend $5 for this lab; generous safety margin above expected ~$0.05 consumption).
23. Save. Anthropic stops accepting new requests from your account once the limit is reached for the cycle.

> **IMPORTANT — SPEND LIMITS ARE A SAFETY NET, NOT A GUARANTEE**
>
> Enforcement is at billing-cycle granularity, not real-time. In-flight requests when the limit hits may still bill. Always pair the cap with active monitoring (Section 6.3).

6.2 Per-request token cap in OpenClaw

maxTokensPerRequest in openclaw.json caps Claude's output per call. This directly bounds per-request cost.

Reference table:

Model	Input ($/1M tok)	Output ($/1M tok)	Recommended max_tokens
claude-haiku-4-5	$0.80	$4.00	1,000 – 2,000 (default for demos)
claude-sonnet-4	$3.00	$15.00	500 – 1,000 (production tasks needing reasoning)
claude-opus-4	$15.00	$75.00	NOT for demos

At maxTokensPerRequest: 2000 on Haiku, each request maxes out at ~$0.008 — well under a cent.

6.3 Monitoring usage

Local log review:

```
cat ~/.openclaw/logs/api.log
```

Console dashboards: **console.anthropic.com → Usage** for token counts, cost trends, per-key breakdowns. Set **billing alerts** under Settings → Notifications to email you at 50% / 80% / 100% of your spend cap.

> **ADVANCED — PROGRAMMATIC MONITORING**
>
> Pipe ~/.openclaw/logs/api.log to a Prometheus exporter (promtail → Loki, or a custom Node script that parses JSON-lines log and exposes a /metrics endpoint). Visualize daily token spend in Grafana. Even a single-VPS deployment justifies this — surprise bills are almost always caused by a runaway loop, and the loop is visible in tokens-per-minute long before it shows up in the next-day Anthropic dashboard.

6.4 Tier 1 rate limits

New accounts start at Tier 1:

Model	Req/min	Input tokens/min	Output tokens/min
claude-haiku-4-5	50	50,000	10,000
claude-sonnet-4	50	30,000	8,000
claude-opus-4	50	500,000	80,000

> **NOTE — AUTOMATIC TIER PROMOTION**
>
> Sustained legitimate usage promotes you Tier 1 → 4 without manual request. Each tier multiplies limits significantly.

7. Safety, Security, and Responsible Use

The habits you set in the demo carry into production. Get them right here.

7.1 API key security

- **Never commit keys.** .gitignore your .env, openclaw.json (if you ever put raw keys in it — don't), and any shell profile fragments containing secrets.
- **Use environment variables, not literals.** Configs reference ${ANTHROPIC_API_KEY}, not sk-ant-...
- **Rotate immediately on exposure.** Console → Delete → Create new → update env vars everywhere.

- **Separate keys per environment.** openclaw-demo-key, openclaw-prod-key, openclaw-staging-key. Revoke one without disrupting the others.
- **Audit periodically.** Console → API Keys → review last-used timestamps. Delete unused keys.

7.2 Local data handling

- All OpenClaw state — configs, logs, cache, task output — lives under ~/.openclaw/. Nothing syncs to OpenClaw's servers.
- The only data leaving your machine is the prompt payload sent to Anthropic. Review Anthropic's privacy policy (anthropic.com/privacy) for how prompt data is handled server-side.
- Do not send PII through this demo — names, emails, phones, addresses, financial data, health records, anything identifying a real person.
- Do not send confidential business data — internal docs, client records, trade secrets, regulated information. The demo environment has no compliance posture.

7.3 Scope limits of the demo

- **No auth on the Gateway.** Port 3000 has no API key, no login, no rate limit beyond Anthropic's own. Never expose port 3000 to the public internet. Don't add a firewall rule or port-forward.
- **Not for production workloads.** Real user data, regulated data (HIPAA / GDPR / PCI-DSS) belongs nowhere near this setup.
- **File and URL allow-lists.** OpenClaw can restrict what directories it reads/writes and what hosts it fetches. openclaw doctor lists current policies. Tighten via ~/.openclaw/openclaw.json under permissions before letting any task that touches files run unattended.
- **Shared machines.** Use the local prefix install (~/.openclaw isolated) and set restrictive file permissions (chmod 600 ~/.openclaw/openclaw.json).

7.4 Responsible AI use

Claude follows Constitutional AI principles and will decline harmful, illegal, or deceptive requests. Build with that in mind:

- Add a human-review checkpoint for any workflow that sends external email, posts publicly, modifies files in shared storage, or takes any action with real-world consequences. AI-generated output going straight to the world without a human in the loop is a risk vector.
- Don't build automations designed to deceive, spam, scrape against terms of service, or evade rate limits.
- Read anthropic.com/usage-policy before going to production. It's short, specific, and binding.

8. Scaling to Production on a VPS

8.1 Why move to a VPS

Local has one architectural limit: it only runs when your laptop is awake. Real automation runs continuously, responds to webhooks, handles scheduled triggers, persists state across reboots. That requires a server.

A VPS gives you:

- **24/7 uptime** for scheduled and event-driven workflows.
- **Root access** to install exactly what you need with no platform-imposed restrictions.
- **Cost predictability** vs. managed cloud — typically 5–10× cheaper for equivalent compute.
- **Data control** — your task content, your logs, your output, your storage policy.

Recommended entry point. Hostinger KVM2 (2 vCPU, 8 GB RAM) — comfortable for OpenClaw + n8n + Postgres simultaneously. Step up to KVM4 if you're running 20+ concurrent workflows or want headroom for memory-heavy embedding tasks.

8.2 Recommended production stack

Component	Tool	Notes
Agent runtime	OpenClaw (self-hosted)	Docker preferred. Register as systemd unit for auto-restart
Visual workflow builder	n8n (self-hosted)	Hostinger hPanel offers one-click n8n template; or npm install -g n8n + systemd unit
AI provider	Claude API (production key)	Separate key from your demo key. Production spend cap sized to monthly budget
Database	Postgres (or SQLite for low-volume)	Persists n8n workflow state, OpenClaw memory, custom data
Reverse proxy + TLS	Nginx + Let's Encrypt (Certbot)	Routes n8n.yourdomain.com → :5678 and openclaw.yourdomain.com → :3000 over HTTPS
Monitoring	Uptime Kuma	Free, self-hosted, dead simple. Alerts on HTTP probe failure

Component	Tool	Notes
Secrets	Doppler, 1Password CLI, or HashiCorp Vault	Don't store production keys in plain env files
Backups	restic → S3-compatible bucket	Encrypted, incremental, scheduled via cron

8.3 High-level VPS setup

A future issue will go deep on each step. The shape:

24. Purchase Hostinger VPS (KVM2 minimum) and provision Ubuntu 22.04 LTS.
25. SSH in. Set up a non-root user with sudo, disable root SSH, enable UFW firewall (allow 22, 80, 443 only).
26. Install OpenClaw via the Linux one-liner.
27. Install n8n via Hostinger's one-click template, or npm install -g n8n then systemd unit.
28. Inject API key via your secrets manager into /etc/environment or the service unit's EnvironmentFile=.
29. Install Nginx, point a subdomain at the VPS, run Certbot for Let's Encrypt TLS, configure reverse proxy blocks for both services.
30. Write systemd units for OpenClaw and n8n with Restart=always, RestartSec=10, and dependency on network-online.target.
31. Set spend limits + billing alerts for the production API key.
32. Deploy Uptime Kuma to monitor both subdomains; configure email/SMS alerting.
33. Set up nightly Postgres dump + restic to off-site backup.

ADVANCED — CONSIDER A MANAGED CONTAINER PLATFORM

If you'd rather skip systemd entirely, deploy OpenClaw and n8n as Docker containers under Coolify or Dokploy (both self-hosted, free, PaaS-style UI on top of Docker). One-click TLS via Traefik, one-click rollback, declarative docker-compose deploys. The trade-off vs. raw systemd is ~500 MB RAM overhead — fine on KVM2 (8 GB).

8.4 Where to learn more

Resource	URL
OpenClaw Documentation	openclaw.ai/docs
Anthropic Developer Console	console.anthropic.com

Resource	URL
Anthropic API Reference	docs.anthropic.com
n8n on Hostinger VPS Tutorial	Hostinger Academy — search "n8n VPS tutorial"
Hostinger VPS Hosting	hostinger.com/vps-hosting
n8n Community Forum	community.n8n.io
Anthropic Usage Policy	anthropic.com/usage-policy
Anthropic Privacy Policy	anthropic.com/privacy

9. Going Further — Advanced Patterns

Once the basics are solid, these are the patterns that separate a lab demo from production-grade AI automation.

9.1 Multi-step agent chains

Replace single tasks with a sequence: Task A's output becomes Task B's input. In OpenClaw, define this as a workflow file with `steps[]` array. Use cases: scraping → cleaning → summarizing → emailing; ticket triage → classification → routing → response drafting.

The pattern that scales: **each step does one thing well** (single responsibility) and **state passes via structured JSON** (not natural language). Natural-language pipes lose precision over more than two hops.

9.2 Retrieval-Augmented Generation (RAG)

Inject your own knowledge base into prompts. Architecture:

34. Embed your documents (chunked) into a vector DB (Chroma, Qdrant, Postgres + pgvector).
35. At task time, embed the user's question, retrieve top-k similar chunks.
36. Stuff retrieved chunks into the Claude prompt as context.
37. Claude answers using your knowledge, not its training data.

OpenClaw doesn't ship a vector store, but a custom task can call your vector DB via HTTP before invoking Claude. The Claude API's prompt-caching feature (cache the system prompt + retrieved chunks for 5 min) cuts RAG cost by ~90% on repeated queries.

9.3 Tool use (function calling)

Claude can invoke "tools" you define — functions, APIs, database queries — instead of guessing. In OpenClaw, expose tools as JSON definitions in the task config. Claude returns a structured `tool_use` block when it wants to call one; your task handler runs the actual function and returns the result back to Claude for the next turn.

Use cases: weather lookup, calendar checks, SQL queries, sending email. Tool use turns Claude from "a smart text generator" into "an agent that takes actions."

9.4 Concurrent task execution

For batch workloads, run multiple Claude calls in parallel. Node-level: `Promise.all([...])` over an array of OpenClaw task invocations. System-level: queue with worker concurrency (BullMQ + Redis is the standard). Respect Tier 1's 50 req/min — exceed it and Anthropic returns `429 Rate Limit Exceeded`.

9.5 Prompt caching

For prompts where the first N tokens are stable (system prompt, knowledge base chunks, few-shot examples) and only the user query changes, mark the stable prefix as cached. Anthropic stores the prefix server-side for 5 minutes. Subsequent requests with the same prefix bypass input tokenization → 90% cost discount on cached portion.

In OpenClaw task JSON, add `cache_control` blocks to the system prompt array. Most cost wins for high-volume tasks come from caching, not model selection.

9.6 Observability beyond logs

Three things to track from day one of production:

38. Token spend per task type. Tag each task with a label; aggregate cost per label daily. Surprise bills almost always trace to one task type running 100× more than expected.
39. Latency p50 / p95 / p99. Claude API latency varies 2–10× depending on input length and model load. Alert on p95 > 8 seconds.
40. Error rates by code. 429 = rate limit (back off); 529 = overloaded (retry with jitter); 500 = transient (retry); 400 = bad request (don't retry, alert).

Wire into existing tooling — Datadog, Honeycomb, Grafana, even a Slack webhook for 503+ rates.

10. Quick Reference Cheat Sheet

Key CLI commands

Command	Purpose
openclaw --version	Confirms CLI install
openclaw doctor	Full diagnostics — Node, Gateway, key, policies
openclaw gateway status	Gateway running state + port
openclaw gateway start	Start the daemon
openclaw gateway install	Register daemon as background service
openclaw onboard	Re-run interactive wizard
openclaw chat "msg"	One-shot message to Claude (test connection)
openclaw run task.json	Execute a workflow task file
cat ~/.openclaw/logs/api.log	Review API call history

Key file paths

File / Directory	macOS / Linux	Windows
Main config	~/.openclaw/openclaw.json	%APPDATA%\openclaw\openclaw.json
API call logs	~/.openclaw/logs/api.log	%APPDATA%\openclaw\logs\api.log
Workflow output	./openclaw-output/ (cwd)	.\openclaw-output\
Shell profile (bash)	~/.bashrc	PowerShell $PROFILE
Shell profile (zsh)	~/.zshrc	N/A

Anthropic Console quick links

Task	Path
Create / manage API keys	Console → API Keys
Set spend limits	Console → Settings → Billing & Usage → Spend Limits

Task	Path
Monitor token usage + cost	Console → Usage
Billing email alerts	Console → Settings → Notifications
Add / update payment method	Console → Settings → Billing & Usage → Payment

Essential settings — demo defaults

Setting	Value
Model	claude-haiku-4-5
maxTokensPerRequest	1,000 – 2,000
Monthly spend cap	$5.00 (Anthropic Console)
API key storage	Env var: export ANTHROPIC_API_KEY=sk-ant-...
Gateway port	3000 (http://localhost:3000)
n8n local port	5678 (http://localhost:5678)
Internet exposure	NEVER for demo — localhost only

You've Built Something Real

A working AI-agent runtime wired to Claude, capable of summarizing, classifying, and chaining tasks — built in an afternoon. A year ago, this took a team of engineers. The path from here is straight: experiment locally until you understand the failure modes, then promote to a VPS with the production stack in Section 8.

Read Section 9 again in two weeks. The advanced patterns become real exactly when you've broken enough things in the demo to understand why they exist.

Questions, bugs, breakthroughs — bring them to the community.

Learn. Grow. Automate.