EN

Practical Guide

OpenClaw: Automating Operations

Crons, Triggers, Task Queues and Escalations

Complete architecture to automate real operations

Mars 2026

Table of Contents

Answering questions is 10% of OpenClaw's value. The remaining 90% lies in automation — workflows that run without human intervention.

1. The Real ROI of OpenClaw: Automation

OpenClaw's operational value isn't measured by the quality of a one-off response. It's measured by tasks that execute without being asked:

• The morning report that arrives before the team is at their desks

• Monitoring that detects an anomaly at 3am and immediately escalates

• An invoice that auto-generates when a project is marked complete

> *"Fundamental rule: any task schedulable without AI should use Linux system cron. AI only intervenes where reasoning is needed."*

2. Crons: Linux System vs OpenClaw Crons

Linux System Cron (Zero API Cost)

Linux system cron executes scripts directly on the server without the OpenClaw runtime. API cost: zero.

Ideal use cases:

• Backups and log rotation

• Data reconciliation (CSV import, database sync)

• Formatted report emails (fixed template, no AI generation)

• Ping monitoring and binary alerts (up/down)

• Automatic workspace Git commits

OpenClaw Crons (API Cost — Use Sparingly)

An OpenClaw cron triggers an agent session — an API call to the LLM with token cost. Justified for:

• Reading and summarizing unstructured content (emails, tickets, logs)

• Contextual decision-making (prioritize, classify, recommend)

• Variable content generation (drafted reports, personalized digests)

• Cross-agent coordination (orchestration)

■ *Rule: OpenClaw crons should not run more often than necessary. A daily digest = 1 session/day. A weekly report = 1 session/week.*

Example System Crontab

```
# Backup workspace Git every hour
0 * * * * cd /workspace && git add -A && git commit -m "auto: hourly checkpoint"

# Clean logs older than 30 days
0 2 * * * find /var/log/app/ -mtime +30 -delete
```

```
# Daily CSV report — deterministic script
30 7 * * * python3 /scripts/gen_daily_csv.py
```

3. Triggers and Events

Heartbeat

The heartbeat wakes the agent at regular intervals for lightweight checks. Good use: check urgent emails 2-3 times a day.

Bad use: heartbeats every 15 minutes to find nothing. An idle heartbeat should return HEARTBEAT_OK with no additional processing.

Inbound Messages as Triggers

A message on the agent's channel is itself a trigger: direct instructions, system notifications (webhooks), messages from other agents.

Scheduled Events in the Workspace

The agent maintains an events/scheduled.json file — a registry of future tasks with their target execution time.

```
# events/scheduled.json
{
"tasks": [
{
"id": "invoice-Q1-2026",
"due": "2026-03-31T09:00:00",
"type": "billing",
"payload": {"client": "Acme Corp"}
}
]
}
```

4. Task Queues

JSON Queue Pattern

A JSON file in the shared workspace acts as a queue. The producer agent adds tasks, the consumer agent processes them and updates their status.

```
# queues/content-queue.json
{
"items": [{
"id": "post-openclaw-b9",
"status": "pending",
"type": "blog_post",
"created_by": "JARVIS",
```

```
"payload": {"title": "Memory and Context"}
}]
}
```

Agent-to-Agent Workflow

→ JARVIS detects deadline → adds task to queues/content-queue.json

→ CYRANO reads queue at next heartbeat → processes → marks status: done

→ JARVIS detects done → publishes to Ghost → notifies via Telegram

5. Escalations

When to Escalate?

• High-impact irreversible decision (key client email, data deletion)

• Request ambiguity (multiple plausible interpretations)

• Unknown situation (case not covered by rules)

• Timeout (task takes too long without result)

• Repeated failure (3 failed attempts)

Escalation Mechanism

```
■■ ESCALATION — Agent LEDGER
Task: Invoice generation Acme Corp Q1-2026
Amount: $8,400
Issue: Hourly rate not confirmed
Options:
[A] Use $125/h (last confirmed rate)
[B] Wait for client confirmation
[C] Suspend — I'll handle manually
Timeout: 4h → action A by default
```

Timeouts and Fallbacks

• Conservative default action: least-risky action if no response

• Task suspension: put on hold with log in queue

• Re-escalation: after N hours, escalate to supervision channel

6. Multi-Agent Orchestration

Lead-Delegate Architecture

Agent JARVIS plays the coordinator role. It doesn't process data itself — it orchestrates:

• Monitors workspace and detects triggers

• Spawns sub-agents with lightweight contexts and precise missions

- Awaits their results (push-based, no polling)
- Aggregates and makes decisions
- Notifies the human when necessary

Context Isolation

Each sub-agent receives a minimal context — only what it needs for its task. Security and cost optimization simultaneously.

```
# outputs/subagent-results/email-digest-20260314.json
{
"agent": "hermes",
"status": "done",
"summary": "23 emails processed, 3 urgent",
"escalations": ["email:overdue-invoice"]
}
```

7. Concrete Automation Patterns

Pattern 1: Daily Email Digest

Agent: HERMES | **Frequency:** 7:45am (OpenClaw cron) | **Cost:** ~2,000 tokens/day

- Reads emails from past 24h via himalaya CLI
- Classifies by priority (urgent / to-do / FYI / newsletter)
- Drafts a structured markdown digest
- Notifies JARVIS → included in 8:15am briefing

Pattern 2: Weekly Report

Agents: LEDGER + JARVIS + CYRANO | **Cost:** ~8,000 tokens/week

- LEDGER compiles weekly timesheets
- JARVIS aggregates infra metrics + git activity
- CYRANO drafts client report
- PDF generated → sent to client

Pattern 3: Automatic Invoicing

Trigger: Project marked "complete" | **Agent:** LEDGER

- Detects status change via file monitoring
- Calculates amount (hours x contractual rate)
- Amount > threshold → human escalation
- If validated: generates PDF invoice and sends

Pattern 4: Infrastructure Monitoring Alert

System cron: every 5 min (zero API cost) | **OpenClaw escalation** if anomaly

• Bash script checks services (ping, HTTP 200, disk space)

• If OK → silent log, zero API cost

• If anomaly → writes to alerts/pending.json

• JARVIS analyzes at next heartbeat → notification or digest

8. Anti-Patterns to Avoid

■ Infinite loops: agent A → agent B → agent A without termination condition

■ Too-frequent heartbeats: 96 sessions/day = 48,000 tokens baseline for nothing

■ Overly broad agents: enormous context, incoherent decisions, high cost

■ Automation without logs: impossible to audit and debug

■ Escalations without timeout: system frozen if human doesn't respond

■ OpenClaw crons for mechanical work: pure waste — system cron suffices

9. Automation Checklist

■ Every scheduled task is classified: system cron or OpenClaw cron (justified)

■ Frequency calibrated to acceptable latency

■ Every workflow has a termination condition

■ Every escalation has explicit timeout and fallback

■ Every automated action is logged (timestamp, agent, decision, result)

■ Sub-agent contexts are minimal

■ Sub-agent results are push-based (no polling)

■ Escalation thresholds documented (amount, impact, irreversibility)

■ Regular workflow tests after every configuration change

■ Monthly token budget estimated for each automated workflow

Conclusion

Automation with OpenClaw isn't a technology question — it's an architecture question. The patterns in this guide run in production. The anti-patterns were all encountered and corrected.

Post 9: Memory and Context — how agents remember, learn, and maintain state between sessions.

Full article: blog.botum.ca/openclaw-automate-operations-crons-triggers-task-queues

Website: www.botum.ca • contact@botum.ca