FR

Guide Pratique

OpenClaw : Automatiser les Operations

Crons, Triggers, Files de Taches et Escalades

Architecture complete pour automatiser des operations reelles

Mars 2026

Sommaire

Repondre a des questions, c'est 10% de la valeur d'OpenClaw. Les 90% restants se trouvent dans l'automatisation — les workflows qui tournent sans intervention humaine.

1. Le vrai ROI d'OpenClaw : l'automatisation

La valeur operationnelle d'OpenClaw ne se mesure pas a la qualite d'une reponse ponctuelle. Elle se mesure au nombre de taches qui s'executent sans qu'on les demande :

• Le rapport matinal qui arrive avant que l'equipe soit au bureau

• Le monitoring qui detecte une anomalie a 3h du matin et escalade

• La facture qui se genere automatiquement quand un projet est marque termine

> *“Regle fondamentale : toute tache qui peut etre planifiee sans IA doit l'etre via cron systeme Linux. L'IA intervient uniquement la ou le raisonnement est necessaire.”*

2. Crons : systeme Linux vs crons OpenClaw

Cron systeme Linux (0 cout API)

Le cron systeme Linux execute des scripts directement sur le serveur, sans passer par le runtime OpenClaw. Cout API : zero.

Cas d'usage ideaux :

• Sauvegardes et rotations de logs

• Rapprochements de donnees (import CSV, sync base de donnees)

• Envoi d'emails de rapport formates (template fixe, sans generation IA)

• Monitoring ping et alertes binaires (up/down)

• Commits Git automatiques du workspace

Crons OpenClaw (cout API — parcimonie obligatoire)

Un cron OpenClaw declenche une session agent — appel API au LLM avec cout en tokens. Justifie pour :

• Lecture et resume de contenu non structure (emails, tickets, logs)

• Prise de decision contextuelle (prioriser, classer, recommander)

• Generation de contenu variable (rapports rediges, digests personnalises)

• Coordination entre agents (orchestration)

■ *Regle : les crons OpenClaw ne doivent pas tourner plus souvent que necessaire. Un digest quotidien = 1 session/jour. Un rapport hebdo = 1 session/semaine.*

Exemple crontab systeme

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

# Nettoyage logs > 30 jours
0 2 * * * find /var/log/app/ -mtime +30 -delete
```

```
# Rapport CSV quotidien — script deterministique
30 7 * * * python3 /scripts/gen_daily_csv.py
```

3. Triggers et evenements

Heartbeat

Le heartbeat reveille l'agent a intervalles reguliers pour des verifications legeres. Bonne utilisation : verifier les emails urgents 2-3 fois par jour.

Mauvaise utilisation : heartbeats toutes les 15 minutes pour ne rien trouver. Un heartbeat inactif doit retourner HEARTBEAT_OK sans traitement supplementaire.

Messages entrants comme declencheurs

Un message recu sur le canal de l'agent est lui-meme un trigger : instructions directes, notifications systeme (webhooks), messages d'autres agents.

Evenements planifies dans le workspace

L'agent maintient un fichier events/scheduled.json — registre de taches futures avec leur date d'execution cible.

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

4. Files de taches (task queues)

Pattern JSON queue

Un fichier JSON dans le workspace partage fait office de queue. L'agent producteur y ajoute des taches, l'agent consommateur les traite et met a jour leur statut.

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

```
"created_by": "JARVIS",
"payload": {"title": "Memoire et contexte"}
}]
}
```

Workflow agent → agent

Exemple BOTUM :

→ JARVIS detecte une deadline → ajoute tache dans queues/content-queue.json

→ CYRANO lit la queue a son prochain heartbeat → traite → marque status: done

→ JARVIS detecte done → publie sur Ghost → notifie via Telegram

5. Escalades

Quand escalader ?

• Decision irreversible a fort impact (email client important, suppression donnees)

• Ambiguite de la demande (plusieurs interpretations plausibles)

• Situation inconnue (cas non couvert par les regles)

• Timeout (tache trop longue sans resultat)

• Erreur repetee (3 tentatives echouees)

Mecanisme d'escalade

```
■■ ESCALADE — Agent LEDGER
Tache : Generation facture Acme Corp Q1-2026
Montant : 8 400 $
Probleme : Taux horaire non confirme
Options :
[A] Utiliser 125$/h (dernier taux confirme)
[B] Attendre confirmation client
[C] Suspendre — je traite manuellement
Timeout : 4h → action A par defaut
```

Timeouts et fallbacks

• Action conservative par defaut : si pas de reponse, action la moins risquee

• Suspension de la tache : mise en attente avec log dans la queue

• Re-escalade : apres N heures, escalade vers canal de supervision

6. Orchestration multi-agents

Architecture chef-delegues

L'agent JARVIS joue le role de coordinateur. Il ne traite pas les donnees lui-meme — il orchestre :

• Surveille le workspace et detecte les declencheurs

• Spawn des sous-agents avec contextes legers et missions precises

• Attend leurs resultats (push-based, pas de polling)

• Agrege et prend des decisions

• Notifie l'humain si necessaire

Isolation des contextes

Chaque sous-agent recoit un contexte minimal — uniquement ce dont il a besoin. Securite et optimisation des couts simultanees.

```
# outputs/subagent-results/email-digest-20260314.json
{
"agent": "hermes",
"status": "done",
"summary": "23 emails traites, 3 urgents",
"escalations": ["email:facture-impayee"]
}
```

7. Patterns d'automatisation concrets

Pattern 1 : Digest email quotidien

Agent : HERMES | **Frequence :** 7h45 (cron OpenClaw) | **Cout :** ~2 000 tokens/jour

• Lit les emails des 24h via himalaya CLI

• Classe par priorite (urgent / a traiter / FYI / newsletter)

• Redige un digest structure en markdown

• Notifie JARVIS → inclusion dans le briefing 8h15

Pattern 2 : Rapport hebdomadaire

Agents : LEDGER + JARVIS + CYRANO | **Cout :** ~8 000 tokens/semaine

• LEDGER compile les timesheets de la semaine

• JARVIS agregge metriques infra + git activity

• CYRANO redige le rapport client

• PDF genere → envoye au client

Pattern 3 : Facturation automatique

Declencheur : Projet marque "termine" | **Agent :** LEDGER

• Detection du changement de statut via monitoring de fichier

• Calcul montant (heures x taux contractuel)

• Montant > seuil → escalade humaine

• Si valide : generation facture PDF et envoi

Pattern 4 : Alerte monitoring infrastructure

Cron systeme : toutes les 5 min (0 cout API) | **Escalade OpenClaw** si anomalie

• Script bash verifie les services (ping, HTTP 200, espace disque)

• Si OK → log silencieux, 0 cout API

• Si anomalie → ecriture dans alerts/pending.json

• JARVIS analyse au prochain heartbeat → notification ou digest

8. Anti-patterns a eviter

■ Boucles infinies : agent A → agent B → agent A sans condition de terminaison

■ Heartbeats trop frequents : 96 sessions/jour = 48 000 tokens de base a rien faire

■ Agents trop larges : contexte enorme, decisions incoherentes, cout eleve

■ Automatisation sans logs : impossible a auditer et debugger

■ Escalades sans timeout : systeme gele si l'humain ne repond pas

■ Crons OpenClaw pour travail mecanique : gaspillage pur — cron systeme suffit

9. Checklist automatisation

■ Chaque tache planifiee est classifiee : cron systeme ou cron OpenClaw (justifie)

■ Frequence calibree a la latence acceptable

■ Chaque workflow a une condition de terminaison

■ Chaque escalade a un timeout et un fallback explicites

■ Chaque action automatique est loggee (timestamp, agent, decision, resultat)

■ Contextes des sous-agents minimaux

■ Resultats des sous-agents en push (pas de polling)

■ Seuils d'escalade documentes (montant, impact, irreversibilite)

■ Tests de workflow reguliers apres chaque changement

■ Budget tokens mensuel estime pour chaque workflow automatise

Conclusion

L'automatisation avec OpenClaw n'est pas une question de technologie — c'est une question d'architecture. Les patterns presentes dans ce guide tournent en production. Les anti-patterns aussi ont tous ete rencontres et corriges.

Billet 9 : Memoire et contexte — comment les agents se souviennent, apprennent et maintiennent leur etat entre les sessions.

Article complet : blog.botum.ca/openclaw-automatiser-operations-crons-triggers-files-taches

Site web : www.botum.ca • contact@botum.ca