EN

Setting Up Your First OpenClaw Agents

JARVIS, HERMÈS and CHRONOS

OpenClaw Series — Post 5/7
BOTUM — Mars 2026

1. Why Specialize Your Agents

A generalist AI agent is like asking one employee to manage infrastructure, emails, the calendar, and copywriting simultaneously. It works… until it doesn't.

A generalist agent accumulates instructions that eventually contradict each other. It juggles heterogeneous contexts in a single context window that saturates. When compaction hits, important information falls out. Behavior becomes unpredictable.

Specialization addresses three concrete problems:

- **Targeted context** — An email agent only loads rules and history related to email. Its context stays lean, relevant, predictable.
- **Defined scope** — The agent can't — by design — take actions outside its domain. The risk of unintended side effects is structurally reduced.
- **Independent maintenance** — You can modify the calendar agent's rules without touching the email agent. Changes are isolated and traceable.

According to an internal BOTUM analysis over 6 months, a network of 5 specialized agents handles 3.4× more tasks per day than an equivalent generalist agent, with an error rate divided by 2.7.

2. Multi-Agent Architecture: Shared Workspace, Distinct Identities

All OpenClaw agents in a deployment share a **common workspace** — a versioned Git directory. It's their common ground: they read from it, write to it, communicate through it. But each agent has a distinct identity.

AGENTS.md — the collective identity card

Describes the complete network: who does what, which agents exist, their responsibilities. Each agent reads it at session start.

SOUL.md — the shared personality

Defines the voice, tone, and values shared by all agents. It's the base contract for the entire network.

Role-specific files

Each agent has its own directory in agents/. It stores its business rules, specific history, and configuration files there.

Shared vs. isolated memory

MEMORY.md and memory/YYYY-MM-DD.md are accessible to all agents. Each agent also maintains isolated memory in its directory.

3. JARVIS Agent — the Infrastructure Guardian

JARVIS is the system agent. Its scope: everything touching infrastructure health, workspace management, and operational continuity of the agent network.

Responsibilities

- Health monitoring — periodic checks of Docker services, critical processes, disk space.
- Automatic Git commits — regular commits to ensure traceability.
- Long-term memory — consolidating memory/ into MEMORY.md.
- Context compactions — triggers clean compaction when agent context windows approach limits.
- Proactive alerts — messaging notifications when critical thresholds are reached.

Typical configuration (agents/jarvis/ROLE.md)

```
# JARVIS — System Agent ## Scope - Docker infrastructure - Git workspace (read/write) - Health monitoring - Context compactions ## Rules - Git commit every 4h if changes present - Messaging alert if service down > 5 min - Consolidate memory/ → MEMORY.md every Sunday 11pm - Never modify other agents' role files ## Crons - 08:00: full infrastructure health check - 20:00: workspace commit + daily log - Sunday 23:00: memory consolidation
```

4. HERMÈS Agent — the Email Pipeline

HERMÈS is the email agent. Its scope: everything that flows through the inbox — reading, classification, digests, standardized replies, conditional forwarding.

Responsibilities

- IMAP inbox reading — secure connection via IMAP/SSL.
- Intelligent classification — client, prospect, partner, newsletter, spam, urgent.
- Daily digests — 8am structured summary: priorities, email count, required actions.
- Business filtering rules — certain emails trigger automatic actions.
- Standardized replies — acknowledgments, confirmations via SMTP.

Typical configuration (agents/hermes/ROLE.md)

```
## Connections - IMAP: retrieved from vault (key: imap-botum) - SMTP: retrieved from vault (key: smtp-botum) ## Classification rules - Subject contains "URGENT" → P1 priority, immediate notification - VIP sender → priority digest - Newsletter/promo → automatic archive - Invoice/payment → notify LEDGER via queue ## Digest — 08:00 daily
```

Core rule: *HERMÈS classifies and delegates — it never usurps other agents' scope. An invoice → task for LEDGER. A publication request → notification to CYRANO.*

5. CHRONOS Agent — the Time Architect

CHRONOS is the calendar agent. Its scope: reading and updating Google Calendar, generating briefings, coordinating reminders between agents.

Responsibilities

- Morning briefing (8:15am) — next 48 hours: meetings, deadlines, potential conflicts.
- On-demand queries — availability, free slots, upcoming events.
- Calendar updates — create, modify, delete on explicit instruction only.
- Reminder coordination — informs other agents of events that concern them.

Typical configuration (agents/chronos/ROLE.md)

```
## Target calendar - Calendar ID: retrieved from vault (key: gcal-primary) ## Morning briefing — 08:15 daily - Window: D+0 to D+2 - Format: meeting list, deadlines, flagged conflicts ## Inter-agent coordination - Publication deadline → notify CYRANO (D-1) - Client meeting → notify NEXUS (D-1) - Invoice due → notify LEDGER (D-2)
```

6. Inter-Agent Communication Protocols

Agents don't communicate in real time. They communicate via **persistent artifacts in the workspace** — JSON files, task queues, shared states.

The task queue

queue/tasks.json is the primary communication channel. A producer agent deposits a task; a consumer agent reads it at its next execution. Each task has an explicit final status (pending, done, failed, skipped).

Shared artifacts

For large data exchanges, the artifacts/ folder is used. HERMÈS deposits a structured inbox export. CHRONOS deposits its daily briefing as JSON.

Triggers

Certain events trigger cascading actions. A trigger in triggers/ contains necessary metadata (slug, language, deadline) without requiring human intervention.

Example task in queue/tasks.json

```
{ "id": "task-2026-03-14-001", "from": "hermes", "to": "ledger", "type": "invoice_received", "priority": "normal", "status": "pending" }
```

7. Errors to Avoid

Overlapping scopes

If two agents both handle "urgent emails," they'll produce two responses to the same email. **Rule: one responsible agent per domain, no exceptions.** Gray areas must be resolved in AGENTS.md.

Memory conflicts

If HERMÈS and JARVIS both write to MEMORY.md without coordination, you get contradictory entries. **Rule: one agent owns long-term memory (JARVIS).** Others deposit notes in memory/YYYY-MM-DD.md.

Infinite task loops

HERMÈS deposits a task for CHRONOS, CHRONOS redepositts for HERMÈS which recreates the same task. **Rule: each task must have an explicit final status** and agents verify a similar task doesn't already exist.

8. A Typical Day with 3 Active Agents

- **07:43** — HERMÈS reads overnight inbox (12 emails). 1 urgent → LEDGER task. Archives newsletters.
- **08:00** — JARVIS checks infrastructure health. All services nominal. Git workspace commit.
- **08:15** — CHRONOS generates briefing: 2 meetings, blog deadline tomorrow. Trigger for CYRANO.
- **09:30** — ARGUS deposits weekly report in artifacts/. JARVIS notifies via messaging.
- **12:00** — HERMÈS processes morning batch (5 emails). 1 standardized reply. 1 task for NEXUS.
- **16:00** — CYRANO, alerted by CHRONOS trigger, drafts the post. Draft in artifacts/drafts/.
- **20:00** — JARVIS final commit. Consolidates memory/2026-03-14.md. Daily operational summary.

No human intervention required over those 12 hours. Each agent operated within its scope, communicated via defined channels, left auditable traces in the Git workspace.

Conclusion

Configuring JARVIS, HERMÈS, and CHRONOS represents the minimum operational foundation for an OpenClaw agent network. These three agents cover the fundamental domains of any organization: infrastructure, communication, and time.

The real difficulty isn't technical — it's organizational. Defining clear scopes, establishing unambiguous communication protocols, and resisting the temptation to add "temporary" responsibilities.

→ Post 6: OpenClaw vs ChatGPT vs Claude API — Which Tool for Which Enterprise Use Case?

■ Read the full post online

This post is part of a 7-article series on OpenClaw deployment in enterprise.

blog.botum.ca/openclaw-configure-first-agents-jarvis-hermes-chronos/