Week 1: From Zero to Four-Node Swarm

February 9, 2026 · by Mark Bartlett

Download this post as PDF

I started the week wanting one thing: Claude Code but local. An AI agent that could research, write, and code — running on my hardware, using my models, with nothing leaving my network.

By the end of the week, I had four nodes talking to each other on my LAN, routing tasks based on GPU capability, and streaming chat responses from a Gemma 3 27B model running on an RTX 3090. Here's how it happened.

The Problem

I've been using Claude Code daily for InsiderLLM — writing articles, researching models, editing templates. It's excellent. It's also $200/month and sends everything through Anthropic's servers.

I have an RTX 3090 sitting on my desk. I have three Lenovo M710Q mini PCs I picked up for $50 each. I have a gigabit switch. I have Ollama. Why can't this stack do what Claude Code does?

Turns out, nobody has really tried — at least not with hardware regular people actually own.

What I Tried First (And Why It Didn't Work)

OpenClaw

OpenClaw is the big open-source AI agent framework. 440K lines of JavaScript. I wrote the setup guide and the security guide for InsiderLLM.

The problem: it's built for cloud-first, single-agent use. Bolting local-first distributed execution onto a 440K-line Node.js monolith isn't a weekend project. And the security model made me uncomfortable — running an AI agent that can execute arbitrary code with full filesystem access is a liability, not a feature.

nanobot

Tried nanobot too. Found a bug where it silently routes traffic through Alibaba's Dashscope API instead of your local Ollama instance. Not malicious — just a routing mistake in the code. But if the whole point is local-first privacy, a bug that sends your prompts to a cloud API defeats the purpose.

The Question

Both tools started from "how do we make a cloud agent?" and tried to add local support later. What if the starting point was different? What if local-first, distributed-first was the foundation?

Building mycoSwarm

I named it after mycelium — the underground fungal network that connects a forest. It doesn't centralize. It finds what's available and connects it. That's what I wanted for hardware.

Day 1-2: Know Your Hardware

Before you can route tasks, you need to know what each machine can do. The first code I wrote was hardware detection:

- GPU detection: what card, how much VRAM, what's loaded
- CPU profiling: cores, speed, available RAM
- Capability classification: GPU inference, CPU inference, CPU worker, storage

Each node announces what it can do. A machine with a 3090 and 24GB VRAM is a GPU inference node. An M710Q with no GPU but 8GB RAM is a CPU worker — good for web scraping, document processing, hosting a vector database.

Day 3-4: Finding Each Other

The nodes need to discover each other without manual configuration. I used mDNS (the same protocol that lets your printer appear on the network automatically). Start a node, and it broadcasts its presence. Other nodes see it and connect.

No central server. No configuration file listing IP addresses. Plug in a machine, start the daemon, and the swarm grows.

Day 5-6: The API and Task Routing

Built a REST API on each node and an orchestrator that routes tasks based on capability and load:

- Chat request comes in → route to the GPU node with the best model loaded
- Research task → route to a CPU worker with internet access
- Embedding request → route to whichever node has the embedding model warm

Streaming responses work end-to-end. Ask a question from any node, the orchestrator finds the right GPU, and tokens stream back in real-time.

Day 7: It Works

Four nodes running:

Node	Hardware	Role
Miu	RTX 3090, 64GB RAM	Executive — Gemma 3 27B, orchestrator
naru	M710Q, 8GB RAM	CPU worker — scraping, ChromaDB
uncho	M710Q, 8GB RAM	CPU worker — document processing
boa	M710Q, 8GB RAM	CPU worker — lightweight tasks

I asked a question from boa (an 8GB machine with no GPU). The orchestrator on Miu received it, ran inference on the 3090, and streamed the response back. boa has no model loaded, no GPU, barely enough RAM for its OS — but it's chatting with a 27B model like it has one.

That's the whole point. The weakest machine in the swarm gets access to the strongest model.

What's Actually Different

The cloud agent model is: one model, one machine, serial execution. Claude Code thinks, then acts, then thinks, then acts. One step at a time.

mycoSwarm can parallelize. When it's working, research happens on one node while drafting happens on another while fact-checking happens on a third. The orchestrator coordinates.

It's early. The orchestrator is basic. The security model needs hardening. The workflow system doesn't exist yet. But the foundation — nodes discovering each other, announcing capabilities, routing tasks, streaming responses across the LAN — that works.

What's Next

Week 2 goals:

- Distributed RAG: embedding model on one node, vector DB on another, chat model on a third
- Persistent memory: conversation history that survives node restarts
- Dashboard: web UI showing cluster status, node health, active tasks
- Security hardening: sandboxed execution, authenticated peer connections

The bigger question: Can this four-node cluster, running on ~$1,000 of used hardware total, match the output quality of a $200/month Claude Code subscription for real-world tasks? Not benchmarks. Real work — articles, code, research.

I don't know yet. But I intend to find out.

---

mycoSwarm is open source. MIT license. If you've got old hardware collecting dust and want to help build distributed local AI, the repo is at github.com/msb-msb/mycoSwarm.

Every node counts.

Source: https://insiderllm.com/blog/week-1-four-node-swarm/