A SETUP GUIDE · CARSON · 2026

Turn Claude Code into a *swarm team.*

Half the tokens. Twice the output. Ruflo orchestrates 60+ specialized agents on top of Claude Code — queens plan, tactical workers ship, and your bill drops thirty to fifty percent on the same work.

AGENTS **60+**

READ TIME	TOKEN SAVINGS	SKILL LEVEL
8 min	30–50%	Beginner+

Contents

Open source. MIT. *#1 on GitHub.* And almost nobody has set it up yet. Ruflo turns a single Claude Code session into a coordinated swarm of 60+ agents — and once it's running, the same work costs you thirty to fifty percent less. This guide gets you from a clean machine to a working swarm in about twelve minutes.

OVERVIEW

What you'll have after this guide

A queen-led agent swarm spawning from inside Claude Code, with smart model routing and cross-session memory wired up automatically.

01 A queen-led tactical swarm spawning from inside Claude Code.

02 Smart 3-tier routing — simple tasks go to free WASM, hard ones to Opus.

03 Cross-session memory via HNSW vector DB. Agents get better every run.

04 A working `/swarm-init` and `/task-create` flow in your terminal.

TOTAL SETUP TIME

12 minutes. From a clean machine — the install is one curl line.

PREREQUISITES

Before you start

Five things have to be in place. Skip any of them and the install fails — sometimes loudly, sometimes silently. The silent ones are worse.

REQUIREMENT	MINIMUM	WHY IT MATTERS
Node.js	20+	Older versions silently fail on the install script.
npm	9+	pnpm and bun also work — npm is just the default path.
Claude Code	Native CLI	`brew install --cask claude-code` (npm install is deprecated as of 2026).
ANTHROPIC_API_KEY	Exported	Without it, init throws "unknown provider" and you'll waste an hour.
PostgreSQL	Optional	Only for RuVector vector DB. SQLite is the default fallback and works fine.

TIP

Put `export ANTHROPIC_API_KEY=sk-ant-...` in your shell **profile** (`~/.zshrc` or `~/.bashrc`), not just the current session — Ruflo's hooks need it on every spawn. Note: Claude Code's subscription login is separate; Ruflo's agent providers still consume the env var.

Multi-provider routing (optional). Add `OPENAI_API_KEY` , `GOOGLE_API_KEY` , `COHERE_API_KEY` , or `OLLAMA_BASE_URL` if you want failover beyond Claude. Pure-Claude is fine for the default 3-tier router (Haiku → Sonnet → Opus).

READ THIS FIRST

Keep `mcp start` running. Always.

The single failure mode that wastes the most first-day time: your terminal closes the MCP server, and you can't figure out why the tools just disappeared from Claude Code.

THE RULE

If Claude Code can't see Ruflo's tools, it's because *mcp start* isn't running. No exceptions. The hammer icon either shows up — or it doesn't, and the server is dead.

Once you've registered Ruflo as an MCP server, Claude Code expects to connect to it on every spawn. If the process isn't running somewhere, the connection fails **silently** — no error, no warning, just no tools.

The fix is dumb but real: open a second terminal, run `npx ruflo@latest mcp start`, and leave it alone. Don't close that window. If you reboot, restart it. Pin the tab.

you're looking for this

```
# in your "mcp" terminal, the server stays running:
> npx ruflo@latest mcp start
MCP server listening on stdio…

# in Claude Code, the hammer icon appears in the input bar.
# if it doesn't, this server isn't running.
```

01

STEP ONE

Install Ruflo

Three paths: the one-line installer (recommended, ~35s), the Claude Code plugin marketplace (no MCP server), or `npm install -g ruflo@latest` if you want the global CLI without piping curl to bash.

A — The fast path

One curl line. Installs the global CLI into `/usr/local/bin`, registers the MCP server, runs diagnostics, and tells you what to do next.

the recommended path

```
> curl -fsSL https://cdn.jsdelivr.net/gh/ruvnet/ruflo@main/scripts/install.sh |
bash

▸ installing ruflo@latest globally…
▸ registering MCP server with Claude Code…
▸ running diagnostics…
✓ ready. run `ruflo init wizard` in your project root.
```

B — Alternative paths

If you don't need explicit swarm control, the **plugin marketplace** route is simpler — but it doesn't auto-register the MCP server, so `swarm_init`, `agent_spawn`, and `memory_store` aren't callable directly. Fine for hook-driven coordination only.

plugin route (no MCP server)

```
> /plugin marketplace add ruvnet/ruflo
> /plugin install ruflo-core@ruflo
> /plugin install ruflo-swarm@ruflo
> /plugin install ruflo-rag-memory@ruflo
```

02

STEP TWO

Initialize and bind to Claude Code

Two commands. The wizard walks you through the project setup; pass `init` without `wizard` to use defaults instead. Then bind it to Claude Code.

A — Initialize the project

From your project root, run the wizard. It asks four questions — TypeScript? Test framework? LLM providers? Memory backend? — then writes `.claude/` and `claude-flow.config.json`.

in your project root

```
> npx ruflo@latest init wizard

? TypeScript or JavaScript? › TypeScript
? Test framework?           › vitest
? LLM providers?            › anthropic
? Memory backend?           › sqlite (default)

✓ wrote .claude/ (40+ skills, hooks, memory)
✓ wrote claude-flow.config.json
```

B — Bind to Claude Code

Register Ruflo as an MCP server. Run this from **inside your project root** — it edits your Claude Code config to bind the server name to the launch command.

MCP register

```
> claude mcp add ruflo -- npx ruflo@latest mcp start
✓ added MCP server "ruflo"
```

03

STEP THREE

Run your first swarm

Spawn a swarm, hand it a task, watch the queen route to the right worker. Or just ask Claude Code naturally — the hooks route through the swarm regardless.

A — Spawn the swarm

Tell Ruflo what you want built. The queen creates three to four tactical workers, initializes shared memory, and waits for tasks. Use `hierarchical` as your default — it gives you a queen-led structure that prevents agent drift. `mesh` is for tightly-coupled work (rare, slow); `adaptive` reconfigures on the fly (use once you have a baseline).

inside Claude Code

```
> /swarm-init --objective "Build a REST API for auth" --topology hierarchical

✓ queen + 3 tactical workers spawned
✓ shared memory: HNSW index online
swarm ready.
```

B — Give the swarm a task

Create a task with a priority. The router picks the most suitable agent. Results go into shared memory so future tasks can reuse the work.

task creation

```
> /task-create --title "Design auth schema" --priority high

▸ routed to: tactical:coder
✓ done in 41s (schema patterns cached)
```

HOW THE BILL DROPS

Why your token bill drops 30–50%

Ruflo's router classifies every task and sends it to the cheapest model that can handle it. The savings come from four stacking techniques — they multiply, not add.

TECHNIQUE	SAVINGS	HOW IT WORKS
ReasoningBank retrieval	−32%	Pulls only the relevant prior patterns into context. Skips stuffing the full history.
Agent Booster (WASM)	−15%	Simple edits like var→const skip the LLM entirely. Pure regex, sub-millisecond.
Pattern caching	−10%	Reuses embeddings across sessions. Second run of similar work is mostly cache hits.
Optimal batching	−20%	Groups related operations into one round-trip instead of N.

The three tiers, in order of cost

Tier 1 — WASM ($0). Simple transforms via Agent Booster. No LLM call at all. **Tier 2 — Haiku / Sonnet ($0.0002–$0.003).** Medium tasks. **Tier 3 — Opus ($0.015).** Complex reasoning, multi-step planning, architecture decisions. The router learns which tier works for each task type over time, and shifts work down a tier when it can.

> *"Convert `var x = 1` to const" — without Ruflo: Sonnet, 2–5 seconds, $0.0003. With Ruflo: Agent Booster detects intent, WASM regex applies it, sub-millisecond, $0.*

COMMON GOTCHAS

Five things that go wrong on day one

TOOLS MISSING IN CLAUDE CODE

Almost always `mcp start` isn't running. Open a side terminal, run `npx ruflo@latest mcp start`, and leave it alone. The hammer icon appears within seconds.

SWARM CONSENSUS IS SLOW

You spawned too many agents. Anything above eight and consensus stalls. Kill the swarm, re-spawn with fewer workers. `--max-workers 4` is a safe default.

"UNKNOWN PROVIDER" ON INIT

`ANTHROPIC_API_KEY` isn't exported. Add `export ANTHROPIC_API_KEY=sk-ant-...` to `~/.zshrc`, source it, re-run `init`.

VECTOR SEARCH RETURNS NOTHING

You skipped `ruflo embeddings init`. Run it from the project root — it builds the HNSW index against your existing memory.

CACHE HIT RATE IS UNDER 60%

Something's misconfigured. Check `npx ruflo memory stats`. Usually the embeddings index is stale or your topology changed mid-project — fix with `ruflo memory rebuild`.

WHAT TO DO NEXT

You're set. Now just use Claude Code.

The hooks system intercepts your edits, routes them through the swarm, and learns in the background. You don't have to do anything different — Ruflo just makes the same work cost less.

1. **Use Claude Code normally.** Ask in plain language. The swarm activates when you need it. `/swarm-init` is only for when you want to spawn explicitly.
2. **Check memory stats occasionally.** `npx ruflo memory stats` — cache hit rate, pattern count, embedding health. Run weekly while you build a baseline.
3. **Try the web UI.** flo.ruv.io — multi-model chat with the same MCP tools. Useful when you're not at a terminal.
4. **Goal planner.** goal.ruv.io — GOAP A* planning for multi-step workflows. Solid for project breakdowns.
5. **Full docs.** The USERGUIDE is 7,600 lines. Worth a skim once: github.com/ruvnet/ruflo/docs/USERGUIDE.md.
6. **Star the repo** if it pays for itself: github.com/ruvnet/ruflo.

flow.

This guide gets DM'd to anyone who drops flow in the comments.

If it ships value for you, the next one ships too.

· MADE WITH CLAUDE CODE · 2026-05-11 ·