EDITION V2 · 2026

InfiniBot Playbook

The 70-Page Playbook for Running Your Business on AI

VOICE AGENTS KANBAN FEDERATION CREDITS REPORTS

CONTENTS · WHAT'S INSIDE

Table of Contents

6	~70	14+
CHAPTERS	PAGES	DIAGRAMS

01 Welcome to InfiniBot GETTING STARTED 3

02 Part 1 — Zero to First Launch FOUNDATIONS 9

03 Part 2 — 10x Your Workflow DAILY DRIVER 26

04 Part 3 — End-to-End Income Loops MONEY MOVES 53

05 Scaling: From One Operator to a Fleet SCALING 74

06 Glossary & Troubleshooting REFERENCE 85

01

PART 01 · GETTING STARTED

Welcome to InfiniBot

> You don't manage agents. You describe outcomes and supervise at the gates.

GETTING STARTED

Welcome to InfiniBot

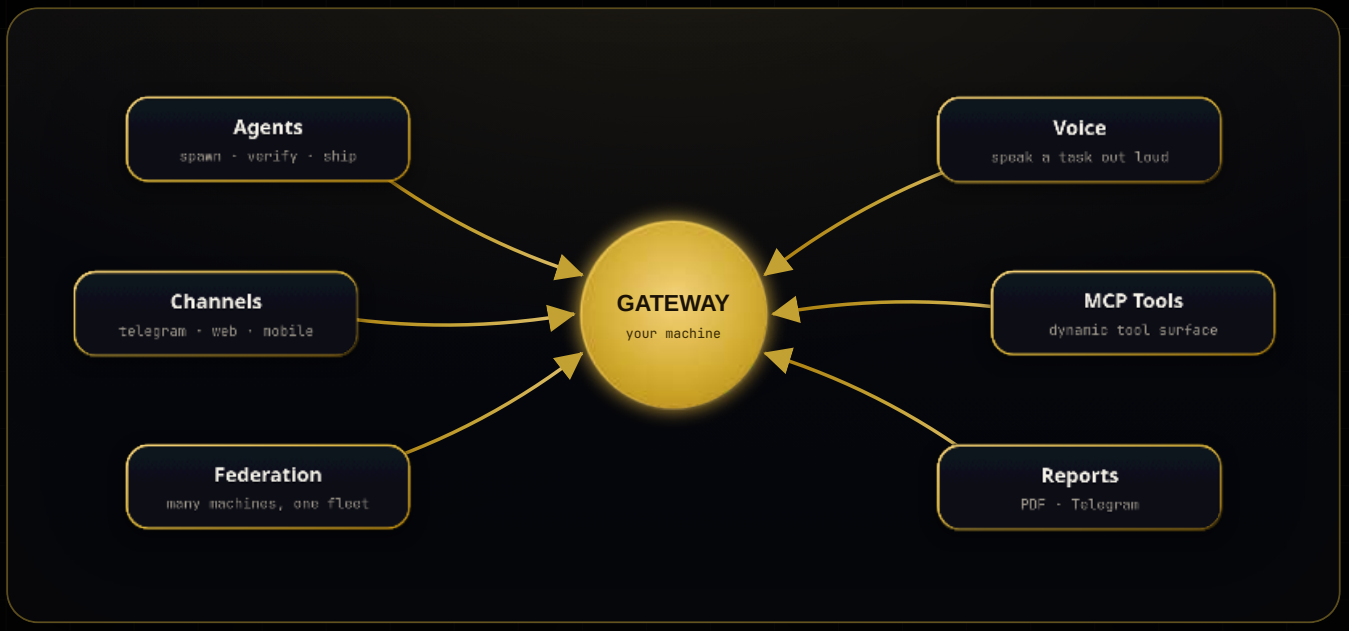


Figure · What is InfiniBot — one gateway on your machine, every capability in orbit.

InfiniBot Playbook — Zero Experience to Income. This is the on-ramp. By the end of this short chapter you'll know exactly what you're holding, who it's for, and how to get a real win on the board in five minutes — before you read another page.

THE INFINIBOT DASHBOARD — YOUR COMMAND CENTER

What this playbook is

InfiniBot is a local-first AI operations platform: a **gateway** that runs on your own machine, a fleet of **agents** that do real work (writing code, drafting content, running research, answering on your channels), and a **control UI** you drive from a browser, the desktop app, or your phone. It is *yours* — your keys, your data, your machine.

This playbook takes you from "I just installed it" to "this is making me money." It is deliberately ordered so you never hit a wall:

- **Part 1 — Foundations.** Install, the first-run wizard, BYOK (bring your own key), and your first conversation. You get the gateway running and talk to your first agent.
- **Part 2 — Daily Driver.** The control UI, projects & kanban, content & scripts, the mobile app, and voice. You learn to *operate* InfiniBot instead of just poking at it.
- **Part 3 — Money Moves.** The income-generating workflows: client work via the CRM, content at scale, productized services, and selling on Whop through KinglyVault.
- **Part 4 — Scaling (this part's companion chapters).** Multi-user profiles, federation across machines, plan-aware credits, and turning one operator into a fleet. Plus the glossary and the troubleshooting playbook that gets you unstuck fast.

You do not need to read it all at once. Read Part 1, get a win, then come back.

Who this is for

The Zero-Experience Beginner

Never opened a terminal, never owned an API key. Every trap that could stop you has a fix in the back of this book.

The Impatient Operator

You already know the tools and just want the shortest path to revenue. Skim the 5-Minute Path, then jump to Part 3.

The Solo Founder

One person, fleet output. You direct agents; they write the code, draft the content, run the research.

The Side-Hustler

Nights and weekends. Run a loop once for your first $100, fifty times for a business — no employees, no inventory.

This book assumes **zero experience**. Not "zero AI experience" — zero. If you have never opened a terminal, never owned an API key, never shipped a thing, you are exactly who this was written for. Every step that *could* trip up a beginner has a matching entry in the Glossary & Troubleshooting chapter at the back.

It is also for the impatient operator who already knows the tools and just wants the highest-leverage path to revenue. If that's you, skim the **5-Minute Success Path** below, then jump straight to **Part 3 — Money Moves**.

You do **not** need to be a programmer. InfiniBot's agents write the code; you direct them. The skill you're building here is *operating a fleet*, not memorizing syntax.

The one idea that makes everything click

> **Bring Your Own Key. The software is the product; the intelligence is *yours* — billed at cost by your own provider, never resold.**
>
> THE ONE IDEA THAT MAKES EVERYTHING CLICK

InfiniBot is **BYOK — Bring Your Own Key**. You plug in your own model provider key (Anthropic, OpenAI, Google, or a local model), and InfiniBot never resells you tokens or inference. The software is the product; the intelligence is *yours*, billed directly by your provider at cost.

That single design choice is why InfiniBot is cheap to run at scale, why your data never leaves your machine unless you send it somewhere, and why "running ten agents at once" is a software decision, not a billing event. Keep this in mind and the rest of the system stops being mysterious.

> **Voice is core, not a bonus.** InfiniBot ships cloud TTS/STT on by default — zero install, works on every OS, BYOK-compatible. You can talk to your fleet from day one. Local voice (Piper/Whisper/Kokoro) is an opt-in upgrade for offline or privacy. Don't skip voice; it's the part people fall in love with.

The 5-Minute Success Path

1
Install & launch
Run the installer or `infinibot onboard`. The UI opens at localhost:18789.

2
Add one key
Paste a single provider key in the wizard. Anthropic is the recommended default.

3
Say hello
Ask your agent to do something tiny and real, and watch it take a real action.

4
Turn on voice
Hit the voice button and ask out loud. Cloud voice is already on — no setup.

5
Bank a real task
Give it work you'd pay a human for. Open Notes — there's the deliverable.

Do these five things in order. Each one is expanded with screenshots in Part 1, but this is the whole arc — and it genuinely takes about five minutes once installed.

1. **Install and launch.** Run the installer (desktop app or `infinibot onboard` in a terminal). The gateway starts and your browser opens to the control UI at `http://localhost:18789`. *(That port matters — see the gotcha in the troubleshooting chapter.)*
2. **Add one key.** In the first-run wizard, paste **one** model provider API key. Anthropic is the recommended default for quality (Claude Opus/Sonnet/Haiku). That's BYOK — you're now wired to real intelligence.
3. **Say hello.** Open the **Chat** view and ask your agent to do something tiny and real: *"List the files in my Downloads folder and tell me what's taking up space."* Watch it actually do it. That moment — an agent taking a real action on your machine — is the click.
4. **Turn on voice.** Hit the voice button and ask the same thing out loud. Cloud voice is already on; no setup. You're now operating hands-free.
5. **Bank a real task.** Give it something you'd actually pay a human to do: *"Draft three cold-outreach emails for a web-design service and save them as a note."* Open the **Notes** view — there they are. That's the loop you'll repeat to make money: *describe the outcome, let the fleet produce it, ship it.*

If all five worked, you have a functioning AI operation running on your own hardware. Everything after this is leverage: more agents, more channels, more clients, more output.

How to read the rest of this book

- **Code, commands, and ports** are shown in `monospace`. Type them exactly.
- **Gotchas** are flagged inline and collected in the back. When something doesn't work, check the Glossary & Troubleshooting chapter *first* — the top 20 beginner traps are all there with fixes.
- **Screenshots** show the real UI, not mockups. If your screen looks different, you may be on a newer build; the concepts hold.

Turn the page. Let's get you installed.

1	0	BYOK	5
COMMAND TO INSTALL	CLOUD LOCK-IN	YOUR KEYS, BILLED AT COST	INCOME LOOPS AHEAD

TRY THIS IN INFINIBOT NEXT CHAPTER →

02

PART 02 · FOUNDATIONS

Part 1 — Zero to First Launch

> One command, one key, and the whole operating system wakes up on your machine.

FOUNDATIONS

InfiniBot Beginner-to-Income Playbook

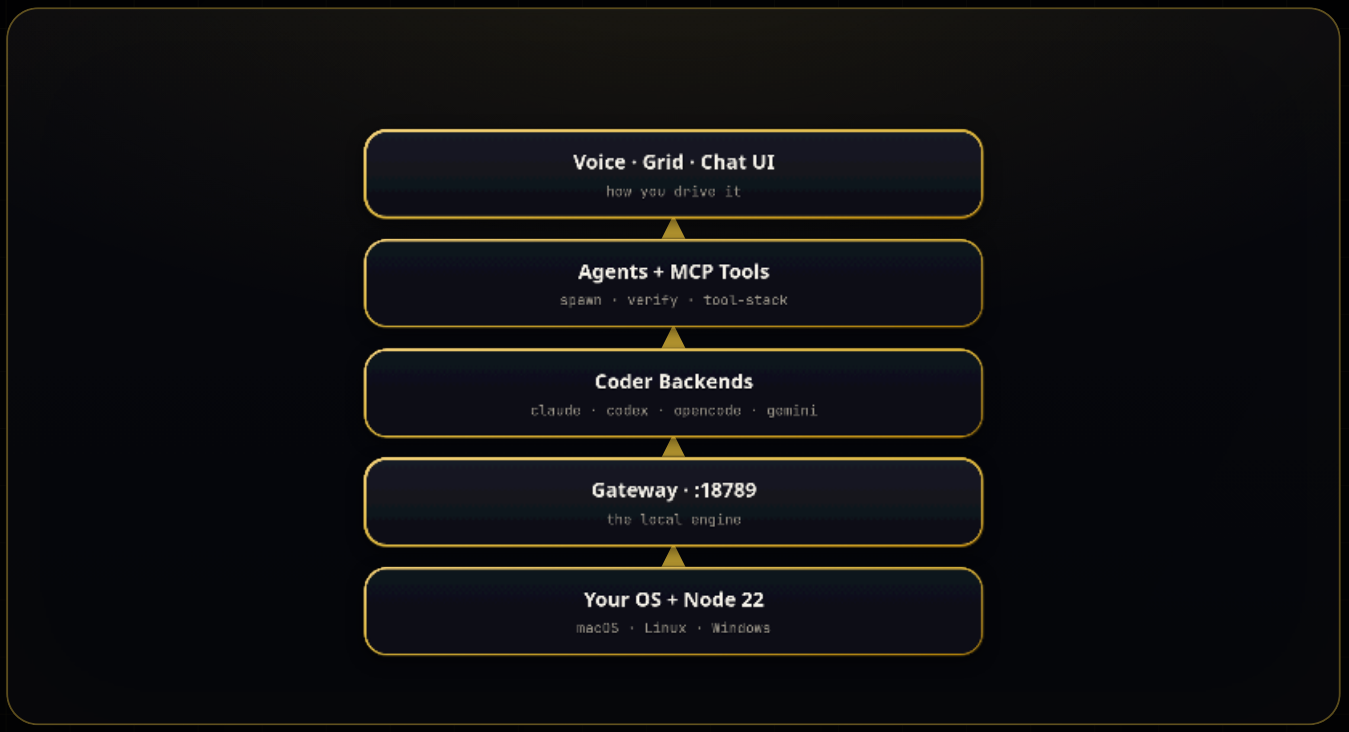


Figure · The InfiniBot stack — one command brings up every layer, bottom to top.

Part 1 — Zero to First Launch

> **Who this is for:** A total beginner. You have never used InfiniBot, maybe never run a terminal command, and you want to get from "nothing installed" to "the app is open and I know what every button does." No experience required. Every term is defined the first time it appears.

What is InfiniBot? (in two sentences)

InfiniBot is a control room for a team of AI agents that live on your own computer — you talk to them by text or voice, hand them tasks, and watch them do real work (writing code, running projects, managing clients) on a live dashboard. You bring your own AI provider key (so you only ever pay the AI company directly, never a markup), and everything runs locally under your control.

Think of it as **mission control for a small army of AI workers** that you own.

---

1. Before you install — what you need

InfiniBot runs on **macOS, Linux, or Windows**. You don't need a powerful machine to start, because the heavy AI "thinking" happens in the cloud (using your provider key).

REQUIREMENT	WHAT IT MEANS IN PLAIN ENGLISH	MINIMUM
Operating system	Mac, Windows, or Linux — any of them work	macOS 12+, Windows 10+, or modern Linux
Node.js 22+	Linux/macOS only — the engine that runs InfiniBot. Free download.	Version **22 or newer**
RAM (memory)	More = more agents at once. 8 GB is fine to start.	8 GB
Microphone (optional)	Only if you want to *talk* to your agents	Any built-in mic

> **Windows users: no Node, no terminal needed.** The ZIP bundle includes everything. Just download, extract, and double-click. See the install table below.

> **Jargon buster — "the cloud":** computers owned by an AI company (like Anthropic) that do the actual thinking. Your laptop just sends them your request and shows the answer.

---

2. Install InfiniBot

InfiniBot is a **member-only** product. Sign in to **KinglyVault → /infinibot** to get your download — you'll see both install options there.

METHOD	BEST FOR	WHAT YOU DO
EXE ZIP bundle	Windows users, zero setup	KinglyVault → /infinibot → download `InfiniBot_v1.0.zip` → extract anywhere → double-click `InfiniBot.exe`
npm install	Linux + macOS (beta)	KinglyVault → /infinibot → copy the one-line npm command → paste into your terminal (requires Node 22+)

> **macOS note:** full signed `.dmg` + menu-bar integration coming soon. For now, macOS uses the same npm path as Linux.

Windows — ZIP bundle (recommended, no terminal required):

1. Sign in at **KinglyVault → /infinibot**
2. Click **Download** `InfiniBot_v1.0.zip`

3. Extract the ZIP to any folder **outside OneDrive** (e.g. `C:\InfiniBot`)
4. Double-click `InfiniBot.exe` — the wizard opens in your browser

> **Jargon buster — "extract":** right-click the ZIP → "Extract All." Windows has this built in — no extra software needed.

Linux / macOS — npm path:

1. Install Node.js 22+ from https://nodejs.org (download + double-click)
2. Sign in at **KinglyVault** → **/infinibot** and copy your install command
3. Paste it into your terminal — it's a single `npm install -g infinibot` line against our gated package source. Re-run any time to update.

When it finishes, start the wizard:

```
infinibot onboard       # the guided first-run setup
```

This opens the **Quickstart wizard** in your browser (covered next). The wizard walks you through everything in steps 3–6 below — you can also reach the same screens later inside the app, which is what the screenshots in this guide show.

---

3. First launch — the gate (password + profile)

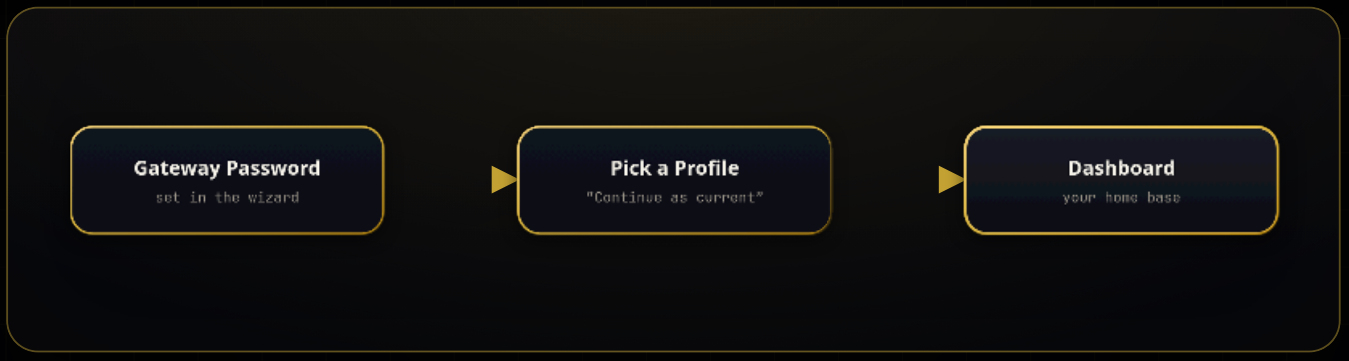
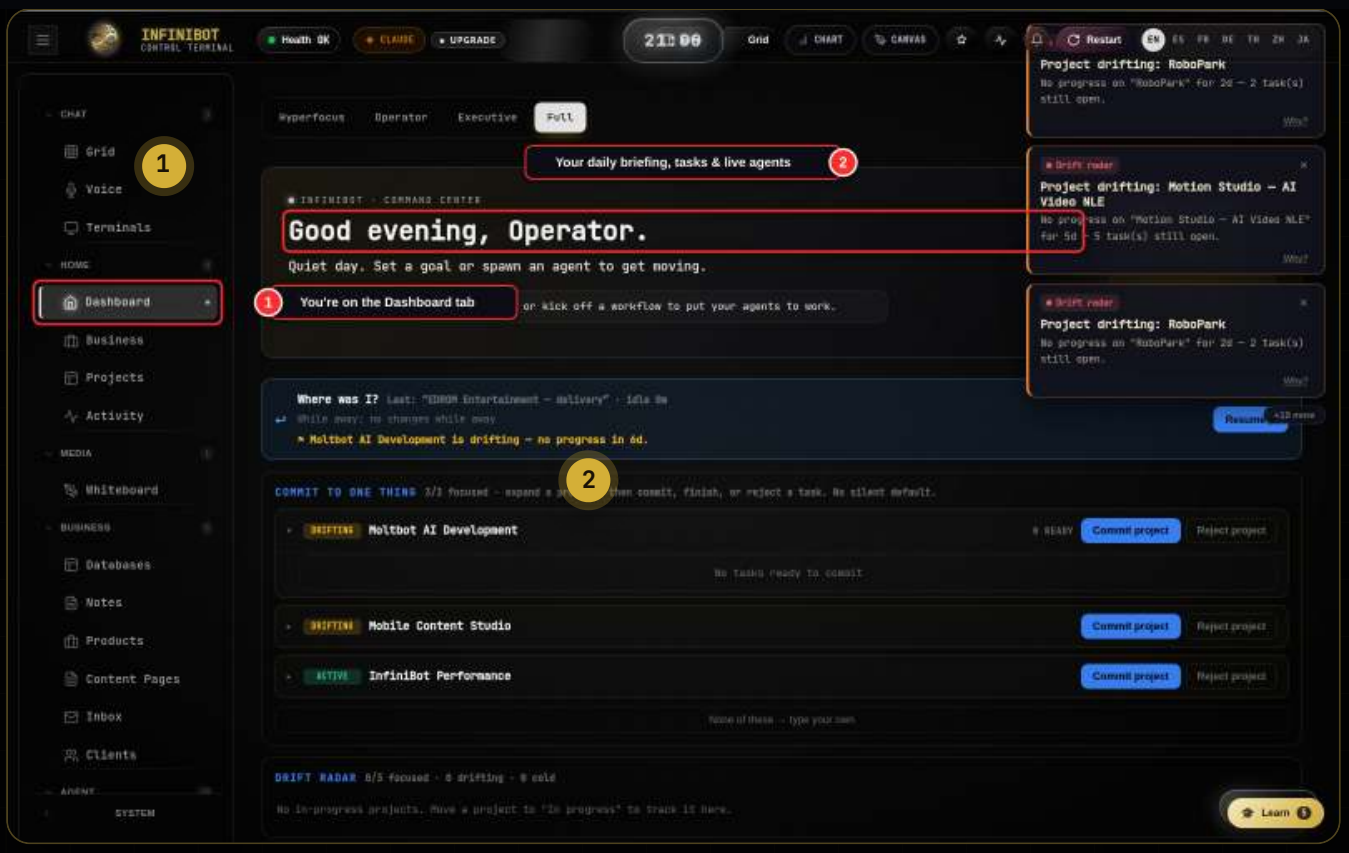


Figure · The gate — password, then profile, then you land on the Dashboard.



PAST THE GATE — THE DASHBOARD, YOUR HOME BASE

> THE GATE
>
> Two quick screens guard the app: the **Gateway Password** you set in the wizard, then **Pick a profile** (click **"Continue as current"**). The gateway serves the UI on **http://localhost:18789**.

The very first time the app loads in your browser (default address **http://localhost:18789**), two quick screens protect it:

1. **Gateway Password** — InfiniBot runs a small local server called the *gateway*. This password makes sure only you can open the control room. You set it during the wizard; type it in here.
2. **Pick a profile** — a *profile* keeps your projects, notes, and chats separate (handy if more than one person uses the same machine). As a beginner, just click **"Continue as current"** to use the default profile.

> **Jargon buster — "the gateway":** the little engine running in the background that the dashboard, voice, and agents all talk to. If the app ever says *"disconnected — retrying"*, the gateway just needs a moment to wake up.

Once you're past these two screens, you land on the **Dashboard** — your home base.

---

4. Bring Your Own Key (BYOK) — connect an AI provider

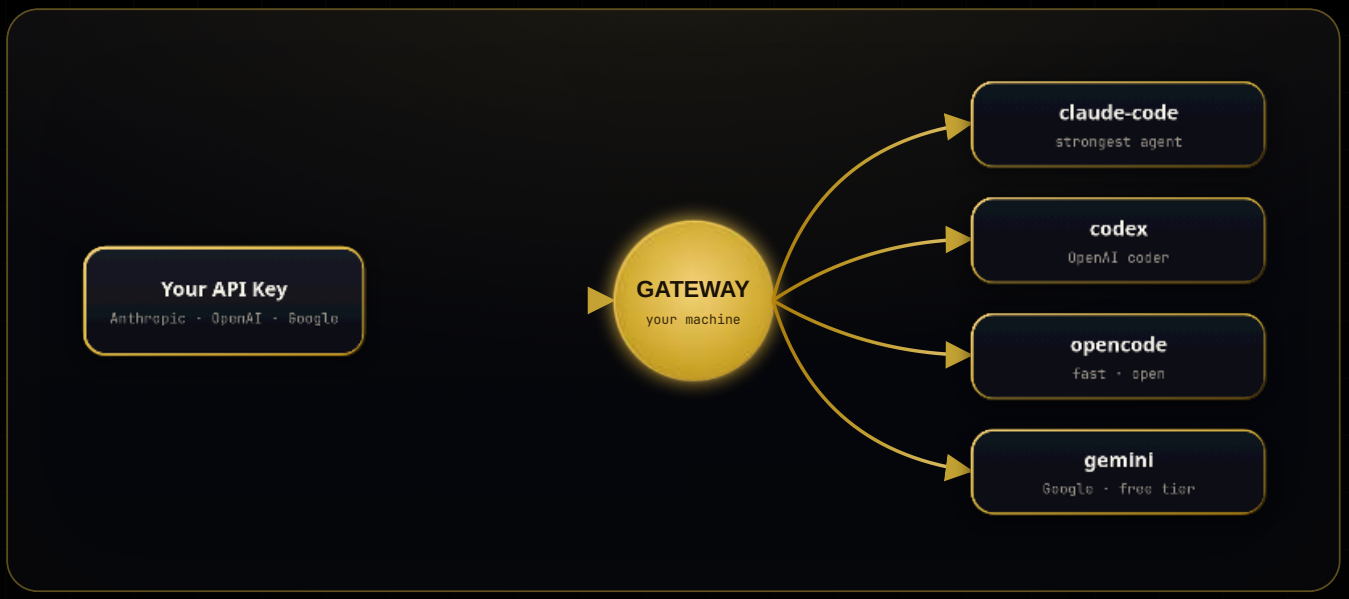


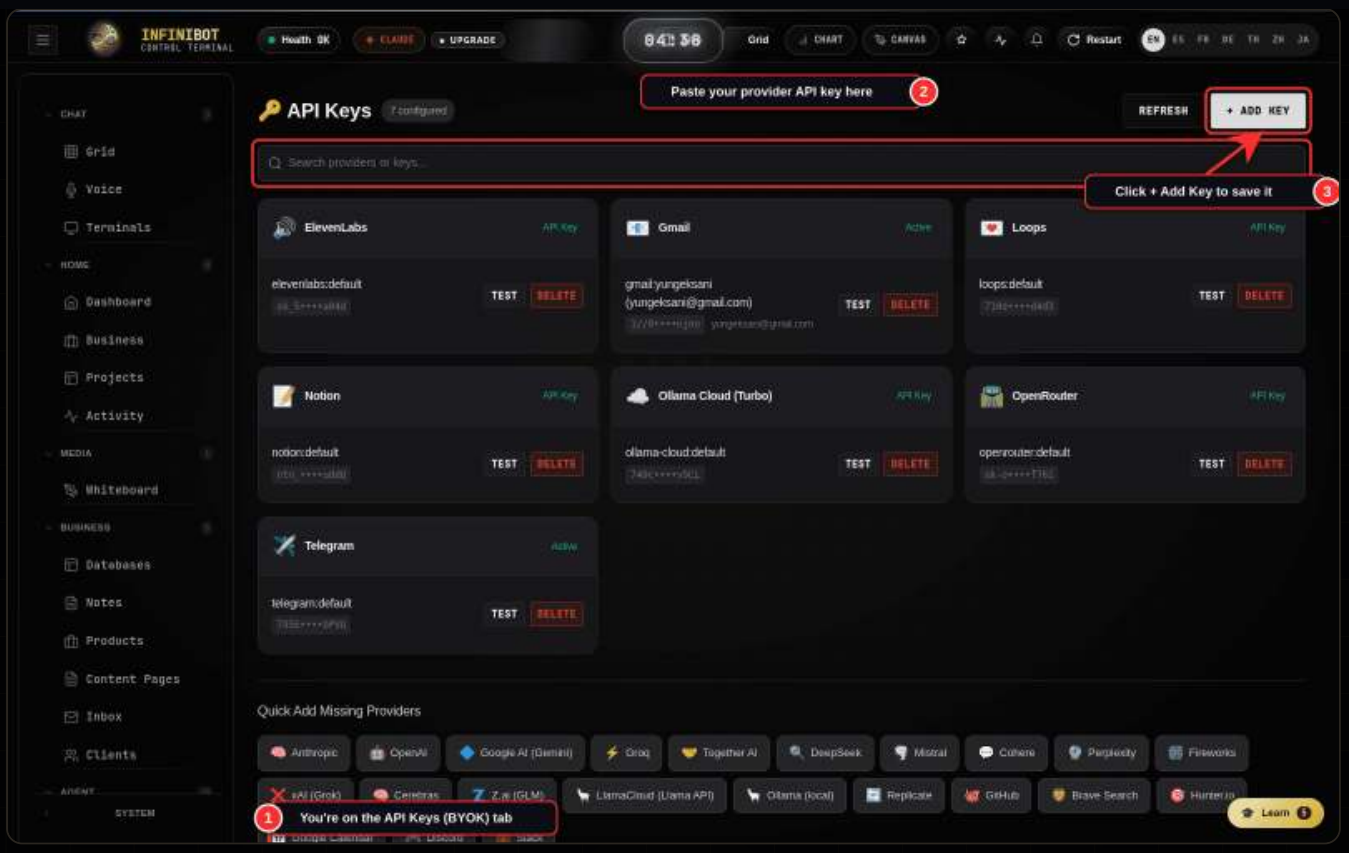
Figure · BYOK — your key flows into the gateway, which fans it out to every coder backend. You pay the provider directly.

◆ InfiniBot never resells AI usage. Your key stays on **your** machine and you pay the provider at **wholesale** — no markup, no lock-in.

InfiniBot doesn't resell AI usage. Instead you paste **your own API key** from an AI company, and you pay that company directly. This is called **BYOK — Bring Your Own Key**. It keeps your costs at wholesale and your data in your control.

> **Jargon buster — "API key":** a long secret password from an AI company (e.g. Anthropic, OpenAI, Google) that lets InfiniBot use your account. Treat it like a credit-card number — never share it.

Open the **API Keys** tab and add a key:



API KEYS (BYOK) TAB WITH THE SEARCH/ADD FIELD AND ADD KEY BUTTON ANNOTATED

1. **You're on the API Keys (BYOK) tab** — the list shows every provider you can connect.
2. **Paste your provider API key** into the field for the provider you have (ElevenLabs, Anthropic/Claude, OpenAI, Google, etc.).
3. **Click "+ Add Key"** to save it. The moment a key is saved, that provider's AI models light up and become usable everywhere in InfiniBot.

Which key should a beginner get? Any one of these gets you started — you only need one:

- **Anthropic (Claude)** — best all-rounder for agents and coding.
- **OpenAI** — popular, widely compatible.
- **Google (Gemini)** — generous free tier to experiment with.

> **Already log in with the Claude/Codex/Gemini desktop CLI?** You may not even need a key — InfiniBot can use that existing login. More on this in the next step.

---

5. Pick a coder backend (who actually writes the code)

CLAUDE-CODE

› Strongest coding agent — great default

CODEX

› OpenAI's coding agent

› Uses Claude (Anthropic)

› Reuses your Claude CLI login — no key to paste

› Uses your OpenAI login / key

› Pick if you prefer GPT models

OPENCODE

› Fast, open option

› Works with many providers

› Good lightweight all-rounder

GEMINI

› Google’s models

› Uses your Gemini login / key

› Generous free tier to experiment

BEGINNER TIP

The backend is a **dropdown you can change per task** — you don’t decide forever. Logged into the Claude CLI? Choose **claude-code** and skip the API-key step entirely.

When you ask an agent to *build* something, it uses a **coder backend** — the underlying coding engine. InfiniBot supports four, and you can switch between them per task:

BACKEND	BEST WHEN…	NOTES
claude-code	You want the strongest coding agent	Uses Claude; great default
opencode	You want a fast, open option	Works with many providers
codex	You prefer OpenAI's coding agent	Uses your OpenAI login/key
gemini	You want Google's models / free tier	Uses your Gemini login/key

You don't have to decide forever — the backend is a dropdown you can change any time. **Beginner tip:** if you installed the Claude CLI and are logged into it, choose **claude-code** and you're done; InfiniBot reuses that login automatically (no key to paste).

Jargon buster — "CLI": a command-line tool you run by typing its name (like `claude`, `codex`, or `gemini`). If you're logged into one of those, InfiniBot can borrow that login so you skip the API-key step entirely.

6. Voice setup — Cloud (default) vs Native (opt-in)

CLOUD · DEFAULT

NATIVE · OPT-IN

- Zero install — nothing to download
- Works on every OS
- Works with your BYOK key
- Instant — the right choice for almost everyone

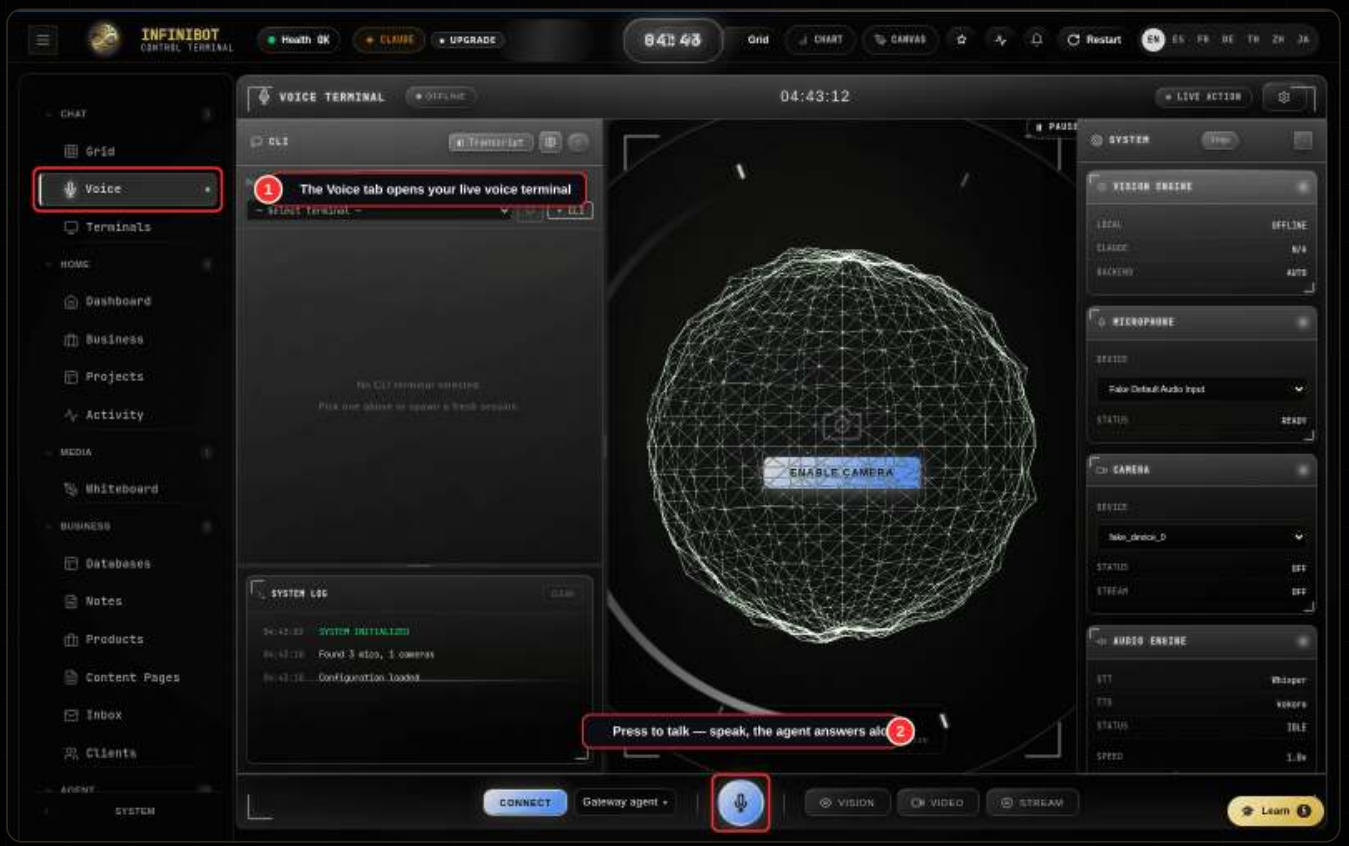
- Runs the voice engine locally
- Offline use + extra privacy
- Downloads larger voice software
- Choose only if you need offline / max privacy

> RULE OF THUMB
>
> Start with **Cloud**. Move to Native later only if you specifically need offline voice. You switch Cloud ↔ Native in the **Quickstart wizard** any time.

Voice is InfiniBot's superpower: you can **talk** to your agents and they talk back. There are two ways to run it.

- **Cloud (the default, recommended for beginners):** zero install, works on every OS, and works with your BYOK key. Nothing to download — just turn it on. This is the right choice for almost everyone starting out.
- **Native (opt-in, advanced):** runs the voice engine locally on your machine for offline use and extra privacy. It downloads larger voice software, so only choose this if you specifically need offline voice or maximum privacy.

You choose Cloud vs Native in the **Quickstart wizard**. After setup, the **Voice** tab is your live voice terminal — press to talk, and your agent responds out loud:



VOICE TAB LIVE TERMINAL WITH THE TALK CONTROL ANNOTATED

1. **The Voice tab** opens your live voice terminal.

2. **Press to talk** — speak your request and the agent answers by voice. (Switch between Cloud and Native back in Quickstart any time.)

> **Rule of thumb:** start with **Cloud**. It's instant, free of extra downloads, and works everywhere. Move to Native later only if you need offline voice.

---

7. The 30-second tour — what every main tab does

Dashboard

Your morning briefing — agent activity, suggested actions, priority tasks, all in one glance.

Grid

A wall of live agent terminals — run several jobs side by side and watch each in real time.

Chat

A focused one-on-one with a single agent. Pick a model, type, watch the reply stream back.

Agents

Your team roster as cards — configure each agent's name, personality, model, and tools.

Projects

A kanban board of project + task cards. Track work by status from Active → Completed.

Discipline

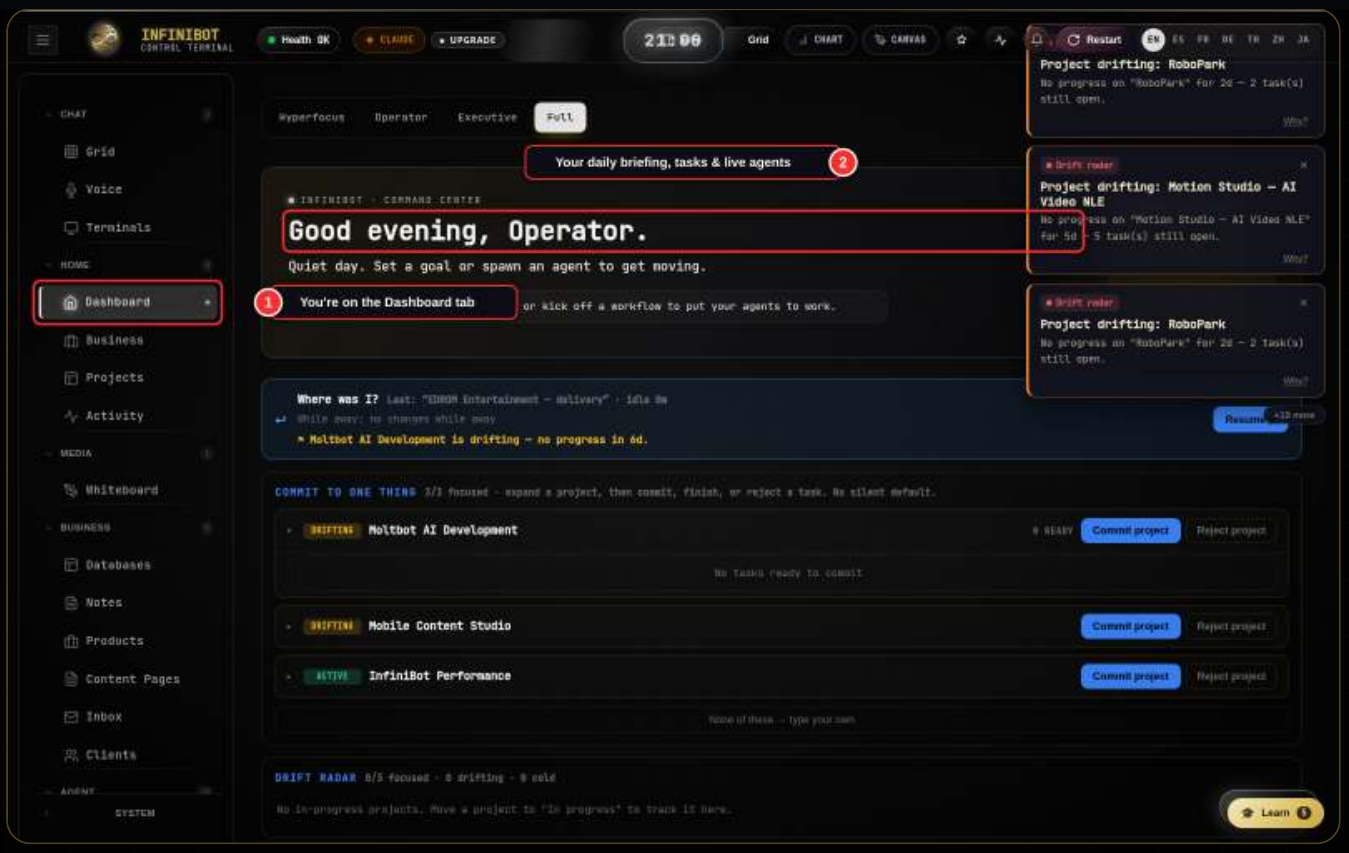
Start your day and commit to one thing; InfiniBot keeps you and your agents on track.

Notes

Your knowledge drawer — capture ideas and briefs your agents can read and pull from.

Credits

Track token usage and provider plan limits per provider — watch your spend, no surprises.



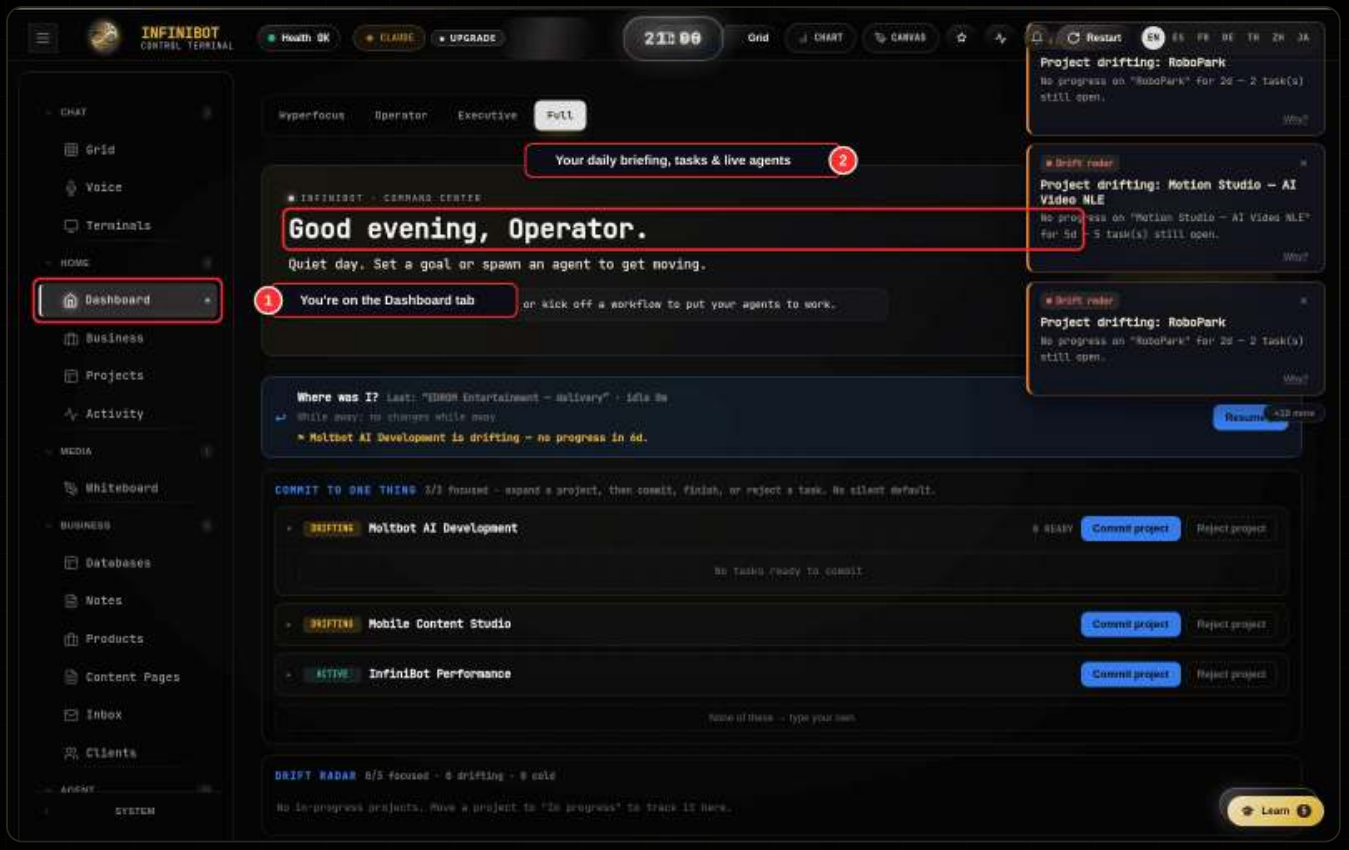
DASHBOARD — START YOUR MORNING HERE

CREDITS — EVERY PROVIDER'S USAGE AT A GLANCE

You're set up. Here's the guided tour. Each screenshot has numbered arrows matching the steps. The main tabs live in the left sidebar.

Dashboard — your home base

Your daily briefing: what your agents are doing, suggested next actions, and priority tasks — all in one glance.



DASHBOARD TAB WITH BRIEFING AREA ANNOTATED

1. **You're on the Dashboard tab.**
2. **Your daily briefing, tasks, and live agents** — start your morning here; it tells you what to focus on and lets you launch work with one click.

Grid — many agents at once

The Grid is a wall of live agent terminals. Each panel is one agent working in real time — perfect for running several jobs side by side.

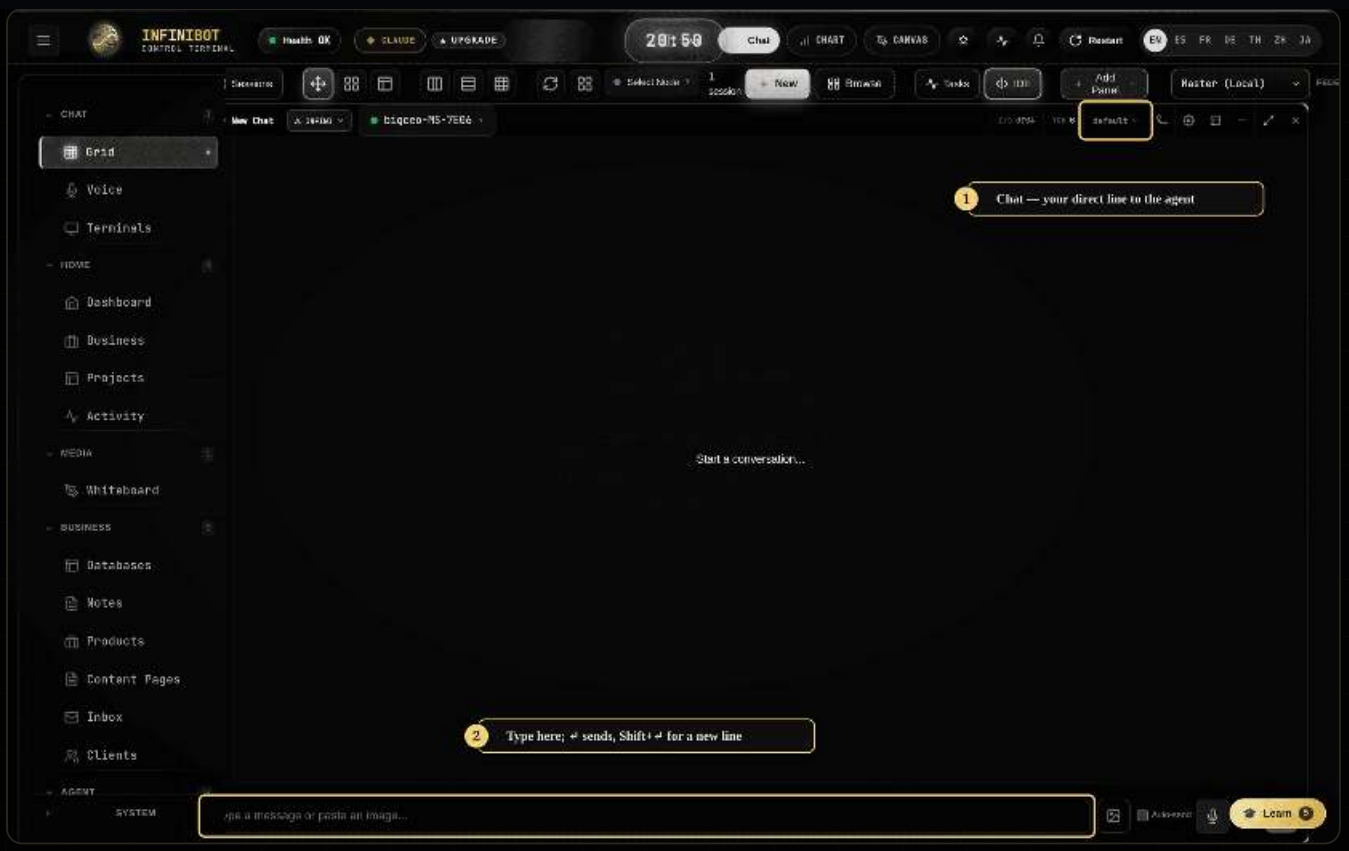
GRID TAB — A FRESH, EMPTY MULTI-SESSION CANVAS WITH THE NEW BUTTON HIGHLIGHTED

1. **You're on the Grid tab.** A fresh install starts empty.
2. **Add a session** — click **New** (top toolbar). Each session you add is a live agent panel you can watch and type into. (Full ten-step Grid walkthrough in Part 2 §1b.)

> **Jargon buster — "terminal":** a text window where you see exactly what the agent is doing, line by line, as it happens.

Chat — talk to one agent

A focused, one-on-one conversation with a single agent — InfiniBot's terminal chat. Pick the agent's model in the header, type in the composer at the bottom, and the reply streams back in the thread. The simplest way to ask for something.

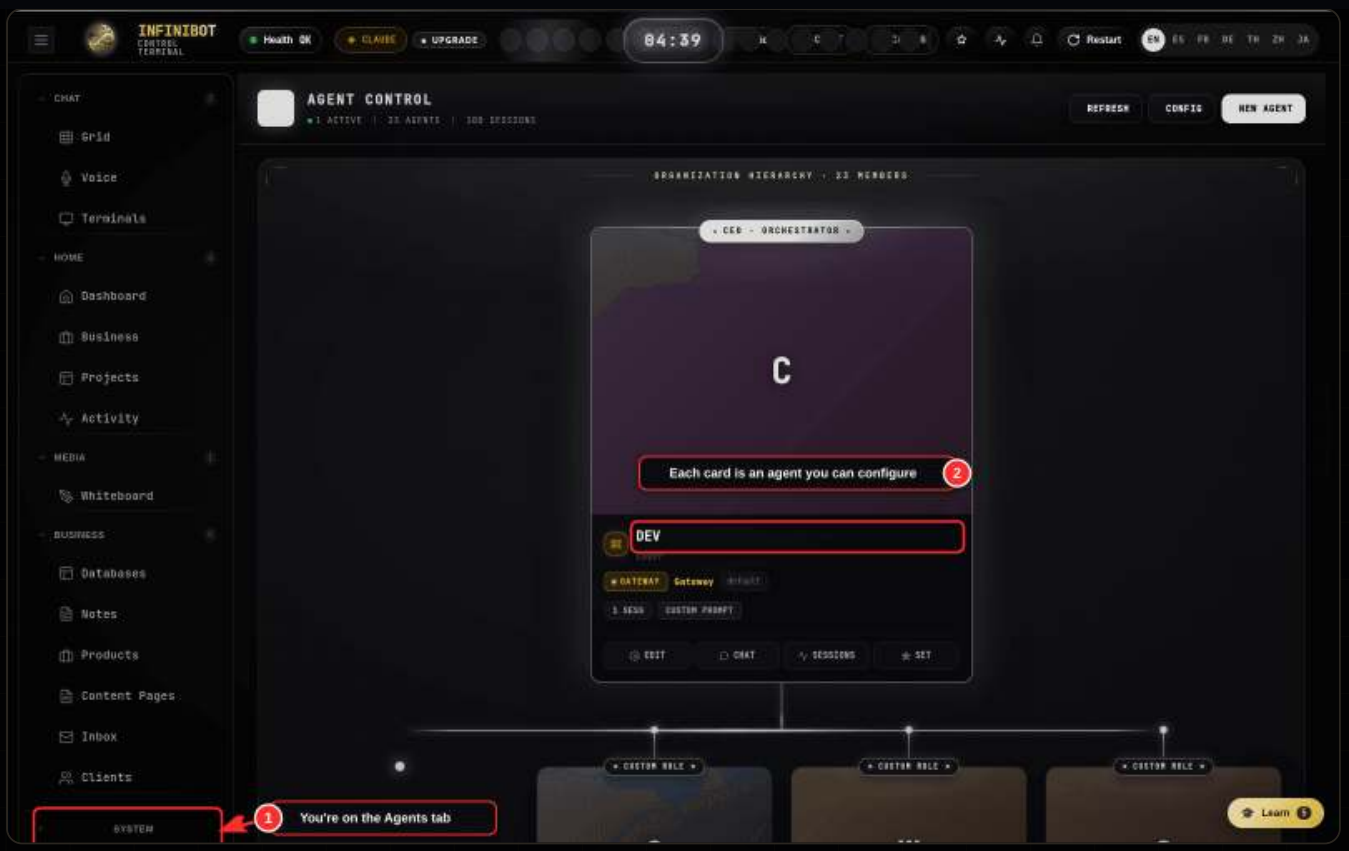


CHAT VIEW — THE TERMINAL CHAT WITH THE MODEL PICKER AND COMPOSER HIGHLIGHTED

1. **Open Chat** from the top bar; the model picker sits in the thread header.
2. **Type a message in the composer** — ask a question or give a task, press ↵, and the agent replies in real time. This is the friendliest place to start. (Full seven-step Chat walkthrough in Part 2 §2.)

Agents — your team roster

Every agent you have, shown as a card. Click one to configure its name, personality, model, and tools.

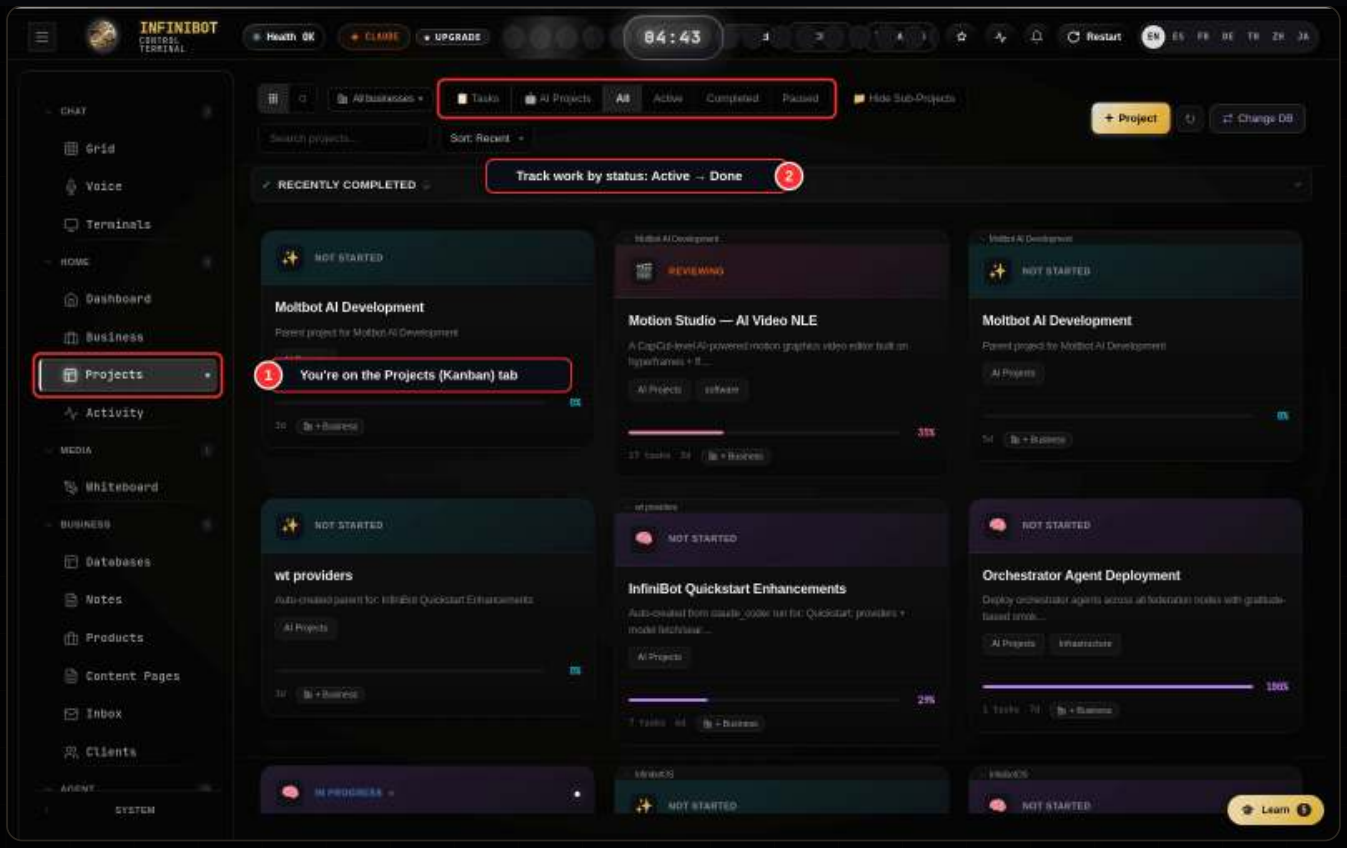


AGENTS TAB WITH AN AGENT CARD ANNOTATED

1. **You're on the Agents tab.**
2. **Each card is an agent you can configure** — give it a role (e.g. "marketing assistant"), pick its model, and turn tools on or off.

Projects — the kanban board

A visual board of project and task cards. Each card carries a **status** — and the filter lanes at the top (**Active** → **Completed**) let you track work as it moves from in-flight to done. Agents and you both update cards as work progresses.



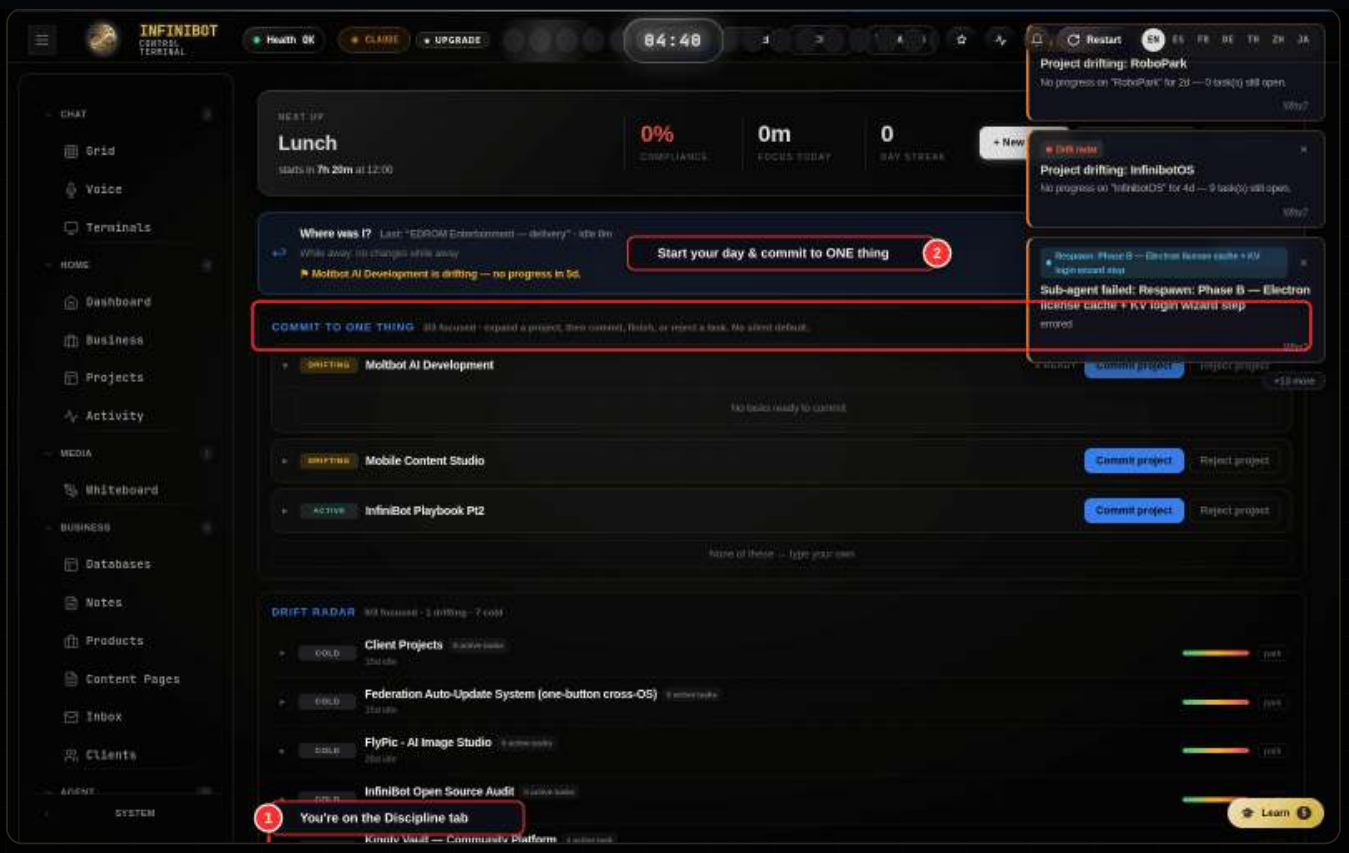
PROJECTS (KANBAN) TAB WITH THE STATUS-FILTER LANES ANNOTATED

1. **You're on the Projects (Kanban) tab.**
2. **Track work by status** — use the filter lanes (Active → Completed) to see what's in flight versus finished; drag cards or let agents update them automatically.

> **Jargon buster — "kanban":** a board of sticky-note cards you track by status. Moving a card toward "done" means "this got more done." It's the standard way teams track progress.

Discipline — start your day & commit

Your focus and accountability hub. Start your day, **commit to one thing**, and InfiniBot keeps you (and your agents) on track instead of drifting.

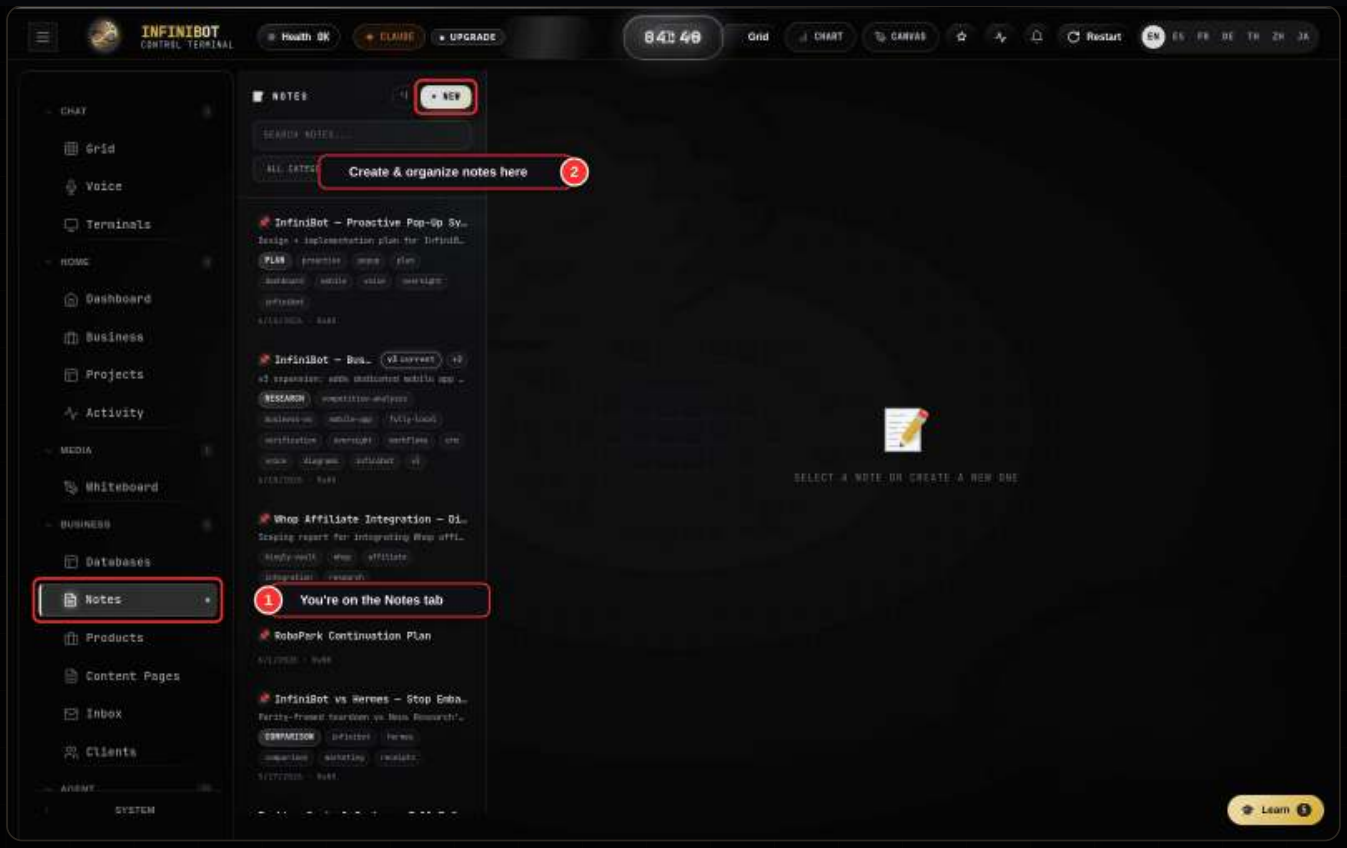


DISCIPLINE TAB WITH THE START-DAY / COMMIT AREA ANNOTATED

1. **You're on the Discipline tab.**
2. **Start your day and commit to one thing** — pick the single most important task; the system gently nudges you and logs progress so the day doesn't slip away.

Notes — your knowledge drawer

A place to capture ideas, briefs, and reference material your agents can also read and use.



NOTES TAB WITH THE NEW-NOTE CONTROL ANNOTATED

1. **You're on the Notes tab.**
2. **Create and organize notes here** — jot down anything; agents can pull from these notes when they work for you.

Credits — track your spend

Because you're using your own provider keys, this page shows exactly how many AI **tokens** you've used and how close you are to each provider's plan limit. No surprises.

CREDITS TAB WITH THE USAGE AND PLAN CARDS ANNOTATED

1. **You're on the Credits tab.**
2. **Track token usage and provider plan limits** — each card is one provider; watch your spend in real time and never get a surprise bill.

> **Jargon buster — "token":** the unit AI companies bill by — roughly ¾ of a word. This page turns tokens into a clear running total so you always know your costs.

You're ready! 🎉

You've installed InfiniBot, gotten past the gate, connected an AI provider with your own key (BYOK), picked a coder backend, turned on voice, and toured every main tab. That's the entire "zero to first launch" journey — you now have a working AI control room.

Quick recap of what each tab is for:

- **Dashboard** — your morning briefing and home base
- **Grid** — many agents working at once
- **Chat** — a simple one-on-one with an agent
- **Agents** — configure your team
- **Projects** — track work on a kanban board
- **Discipline** — start your day, commit to one thing
- **Notes** — capture knowledge agents can use

- **Credits** — watch your AI spend
- **Voice** — talk to your agents out loud
- **API Keys** — manage your BYOK provider keys

→ Continue to Part 2

In **Part 2 — Your First Real Workflow**, you'll put this to work: spawn your first agent, hand it a real task, watch it on the Grid, and turn the result into your first piece of income-producing output. *See* `docs/playbook/02-*.md`.

Screenshots in this guide were captured from a live InfiniBot gateway and annotated with the exact buttons and fields referenced in each step.

BASH · LINUX / MACOS

```bash
# Step 1 — sign in at KinglyVault → /infinibot
# Step 2 — copy your install command (member-only gated package)

# paste the one-liner from your member page — example:
npm install -g infinibot --registry https://pkg.kinglyvault.com

# start the guided first-run wizard
infinibot onboard
```

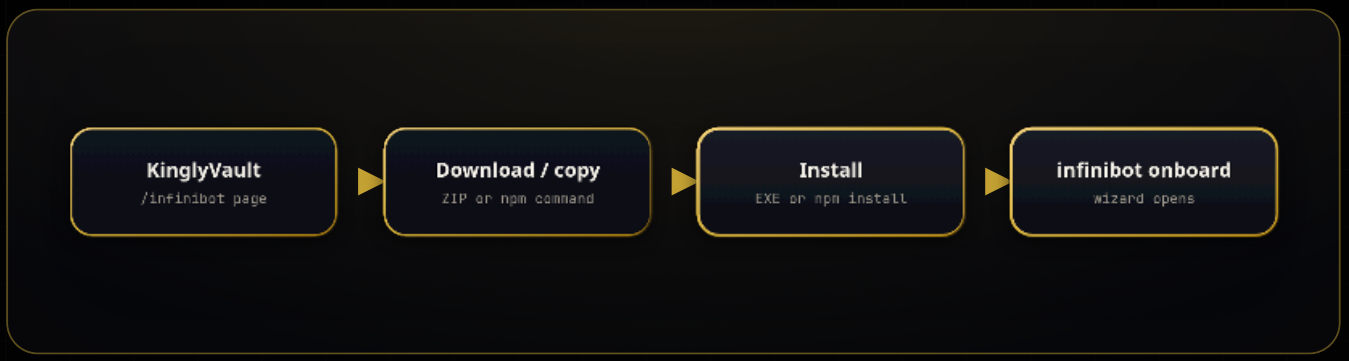


Figure · The install path — KinglyVault /infinibot → download/copy → install → onboard wizard.

TRY THIS IN INFINIBOT NEXT CHAPTER →

03

PART 03 · DAILY DRIVER

Part 2 — 10x Your Workflow

> Speak the task once. The fleet spawns, works, and reports back while you move on.

DAILY DRIVER

InfiniBot Beginner-to-Income Playbook

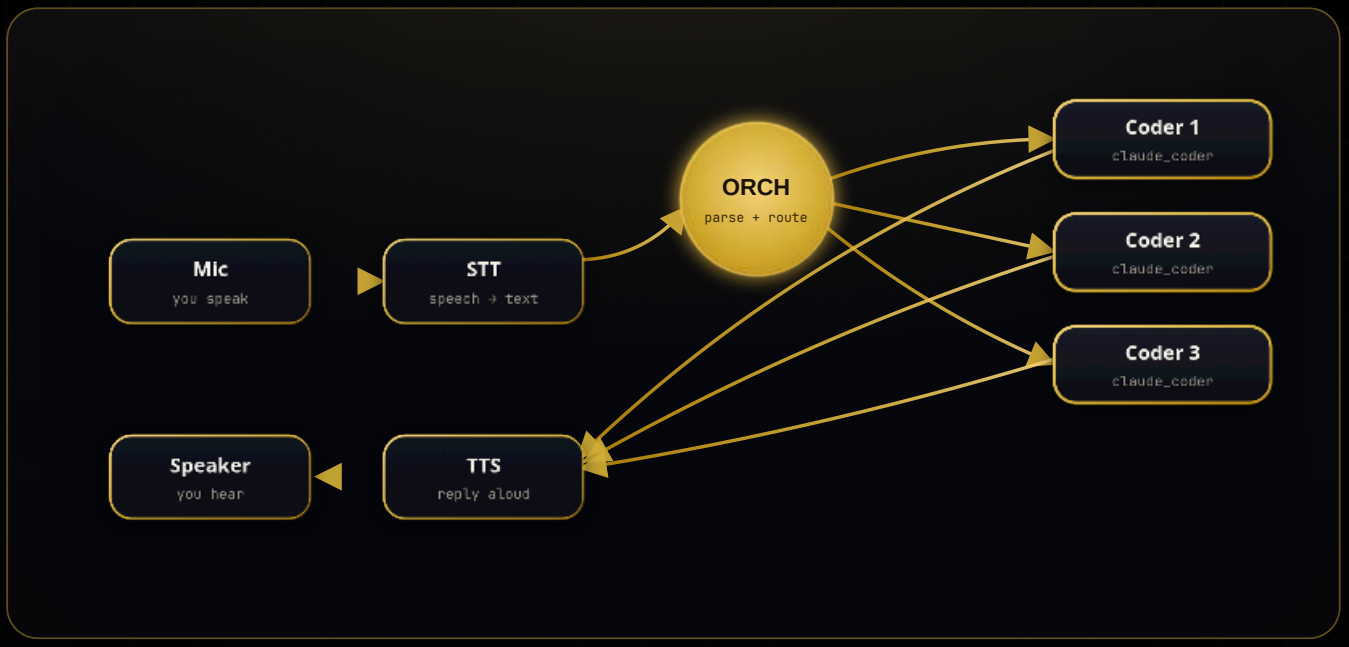


Figure · The voice loop — speak once, the orchestrator parses intent and fans out parallel coders, then reports back aloud.

Part 2 — 10x Your Workflow

> Part 1 got you installed and oriented. Part 2 is about **throughput**: turning one operator into a fleet. Every pattern below is a real surface in the running control terminal (`http://localhost:18789`) — each screenshot is a live capture with the exact controls circled. Work through them in order; by the end you'll be dispatching parallel agents by voice, queuing work on a kanban board, looping background workflows, and watching the whole operation move on a world map.

The shape of every section is the same:

Goal → Steps → Screenshot (with arrows) → Time saved vs. doing it by hand.

How to read the screenshots

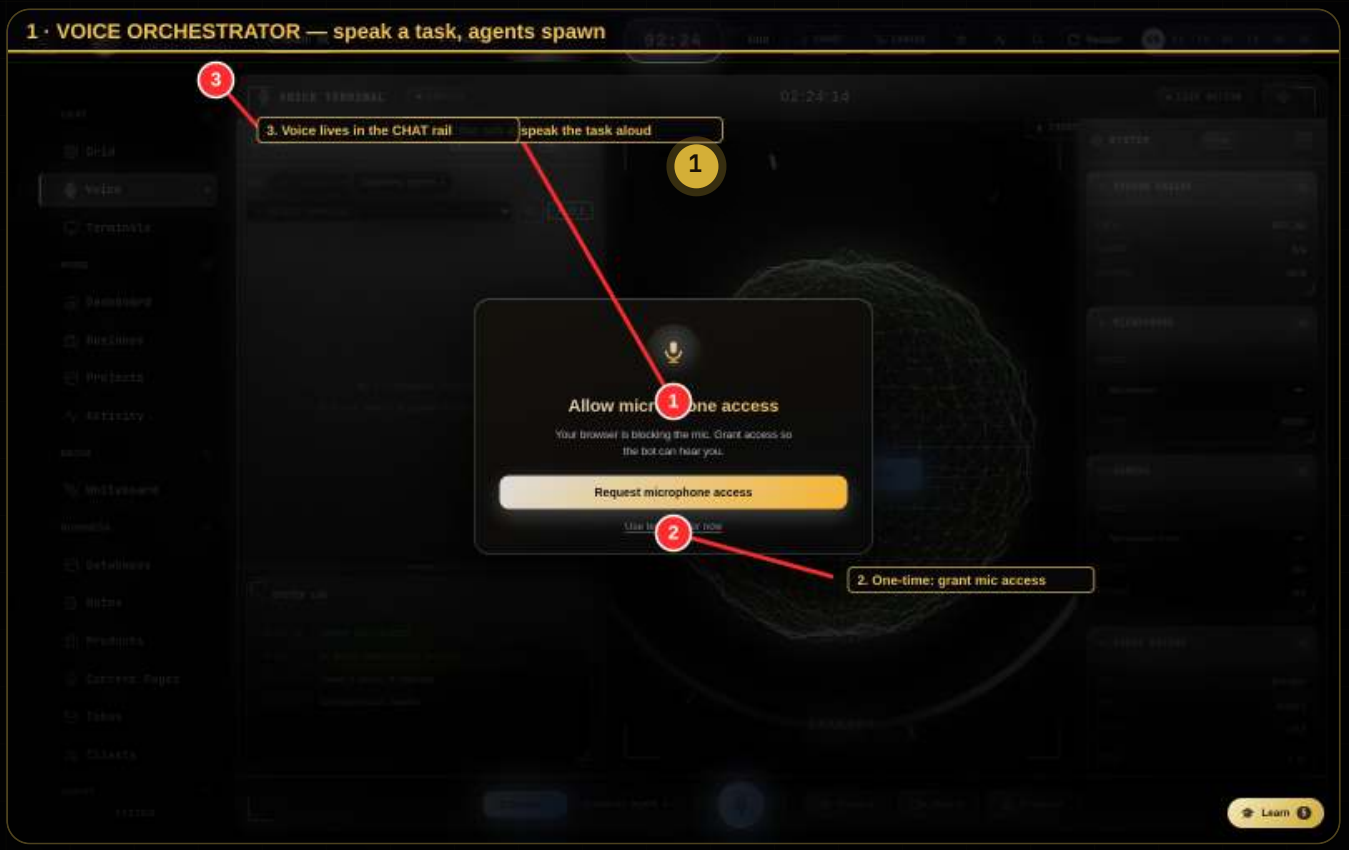
Every image is annotated **onto the pixels** — gold arrows, numbered gold circles, and gold labels. Each arrow tip lands on the exact control it names. Follow the numbers in order; the numbers in the prose match the circles in the picture. The Grid and Chat walkthroughs below are **multi-step**: one screenshot per click, so you can reproduce every state on your own machine.

---

1 · Voice Orchestrator — speak a task, agents spawn

1	N	~10s	~95%
SPOKEN SENTENCE	AGENTS SPAWNED	TO DISPATCH	LESS TYPING

◆ Say it once; the orchestrator turns one sentence into **parallel** `claude_coder` spawns that run at the same time — not back-to-back.

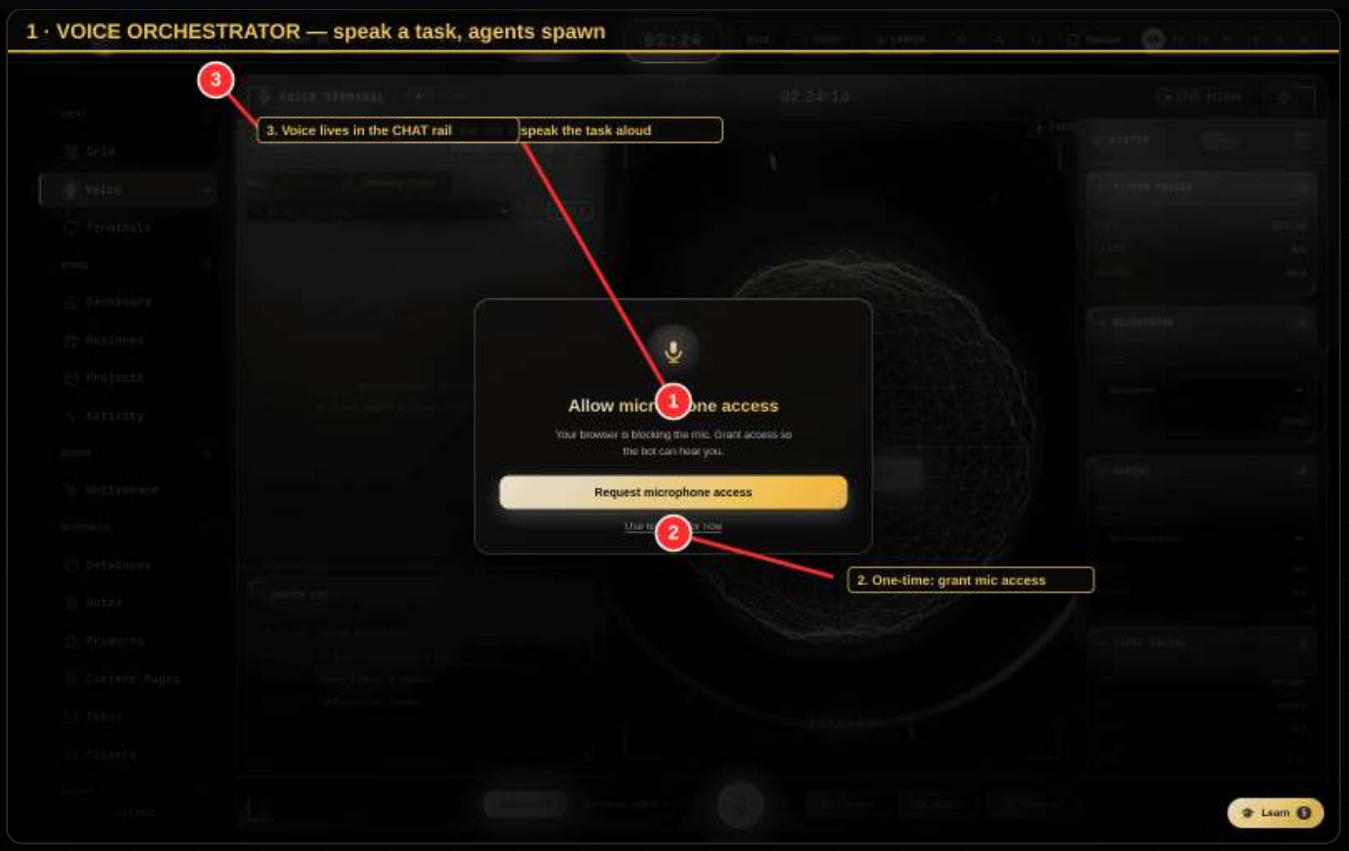


THE VOICE ORCHESTRATOR — SPEAK A MULTI-AGENT DISPATCH

Goal: Stop typing prompts. Say what you want out loud; InfiniBot's voice orchestrator parses it and spawns the right cloud coders to do it.

Steps:

1. Open the **Voice** terminal from the CHAT rail in the left nav.
2. Grant microphone access once (browser one-time prompt).
3. Click the orb and speak your task — e.g. *"Spin up two coders: one to fix the login bug, one to write tests for the checkout flow."*
4. The orchestrator (single-source prompt in `src/agents/voice-orchestrator-prompt.ts`) turns your sentence into one or more `claude_coder` spawns. Pick the CLI backend (Claude / Codex / OpenCode / Gemini) from the header selector before you talk if you want a specific one.



VOICE ORCHESTRATOR

Time saved: Dictating a multi-agent dispatch takes ~~10 seconds. Hand-writing two well-scoped prompts and launching them from a terminal is ~~ 3–4 minutes. And because the orchestrator spawns them in ~~**parallel**, the two jobs run at once instead of back-to-back. ~~**95% on dispatch, and 2× on execution.**

1b · The Grid — a guided, click-by-click walkthrough

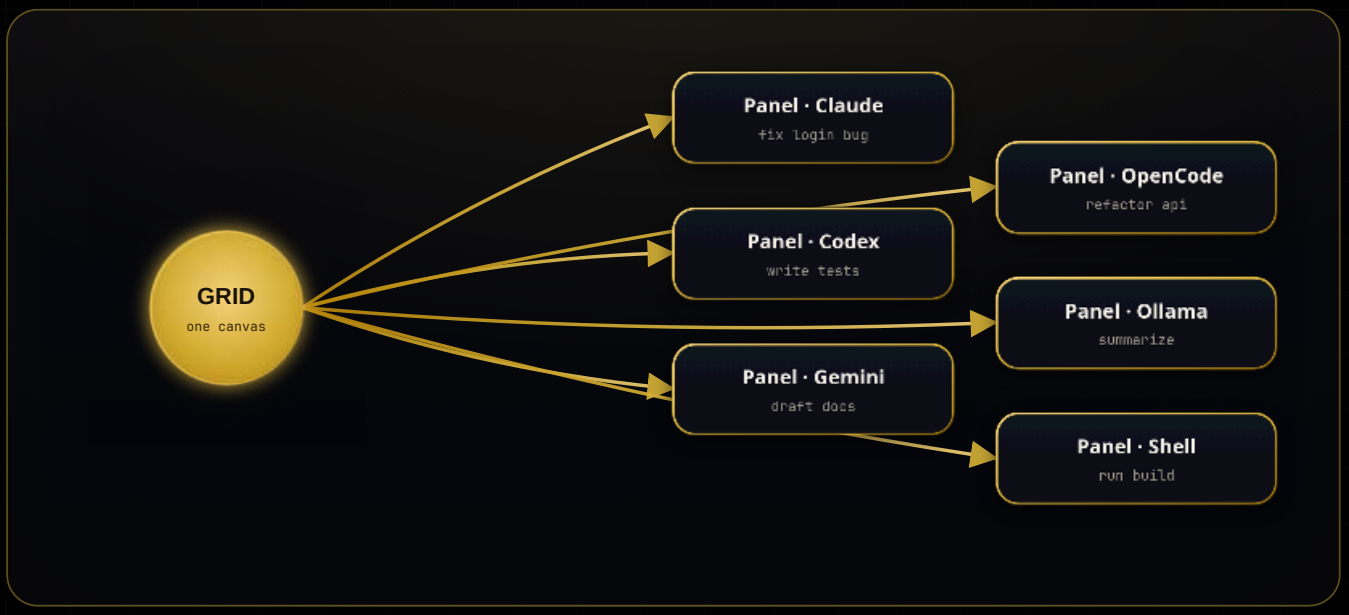
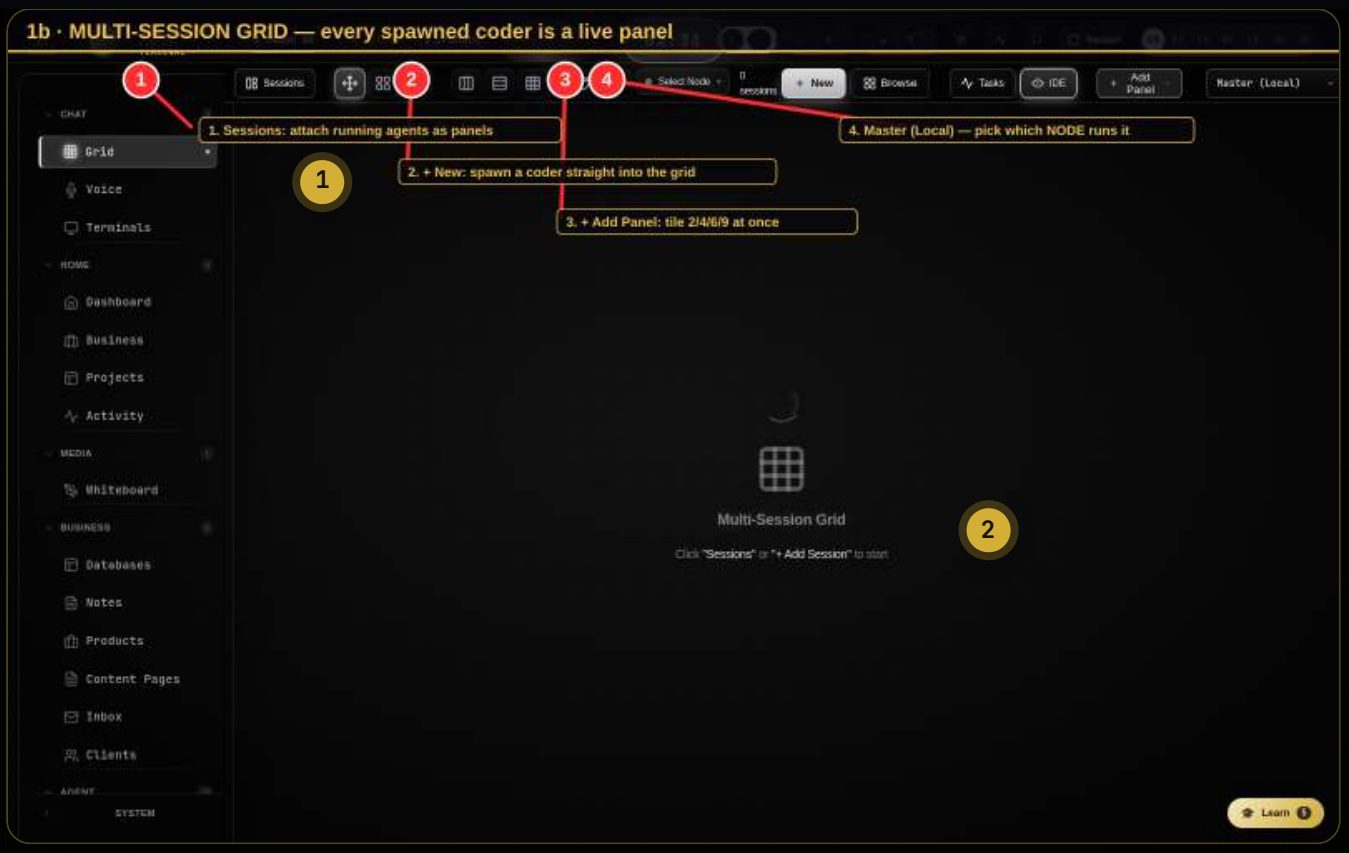


Figure · The Grid — one canvas fans into many live session panels, each with its own model, chat and transcript.

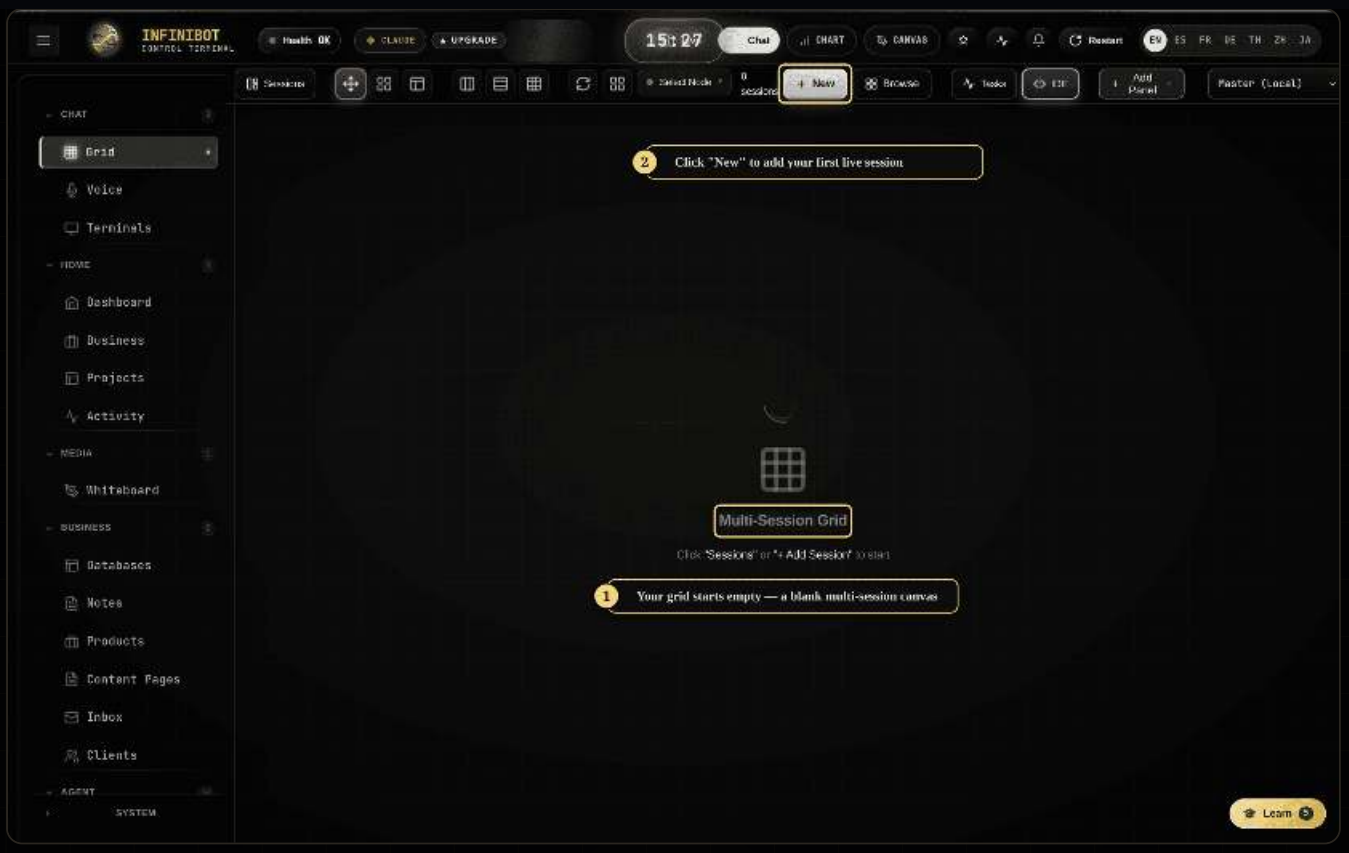


THE GRID — A THREE-AGENT FLEET ON ONE SCREEN, EACH PANEL INDEPENDENTLY DRIVEABLE

Goal: Watch and steer many agents at once instead of tabbing between terminals.

The Grid (`/grid`) is a **multi-session canvas**: every agent is a live panel with its own model, its own chat, its own transcript. Below is the exact sequence — ten clicks, ten screenshots — from an empty grid to a three-agent fleet.

① **Start empty.** A fresh grid is a blank canvas. The `New` button (top toolbar) adds your first live session.

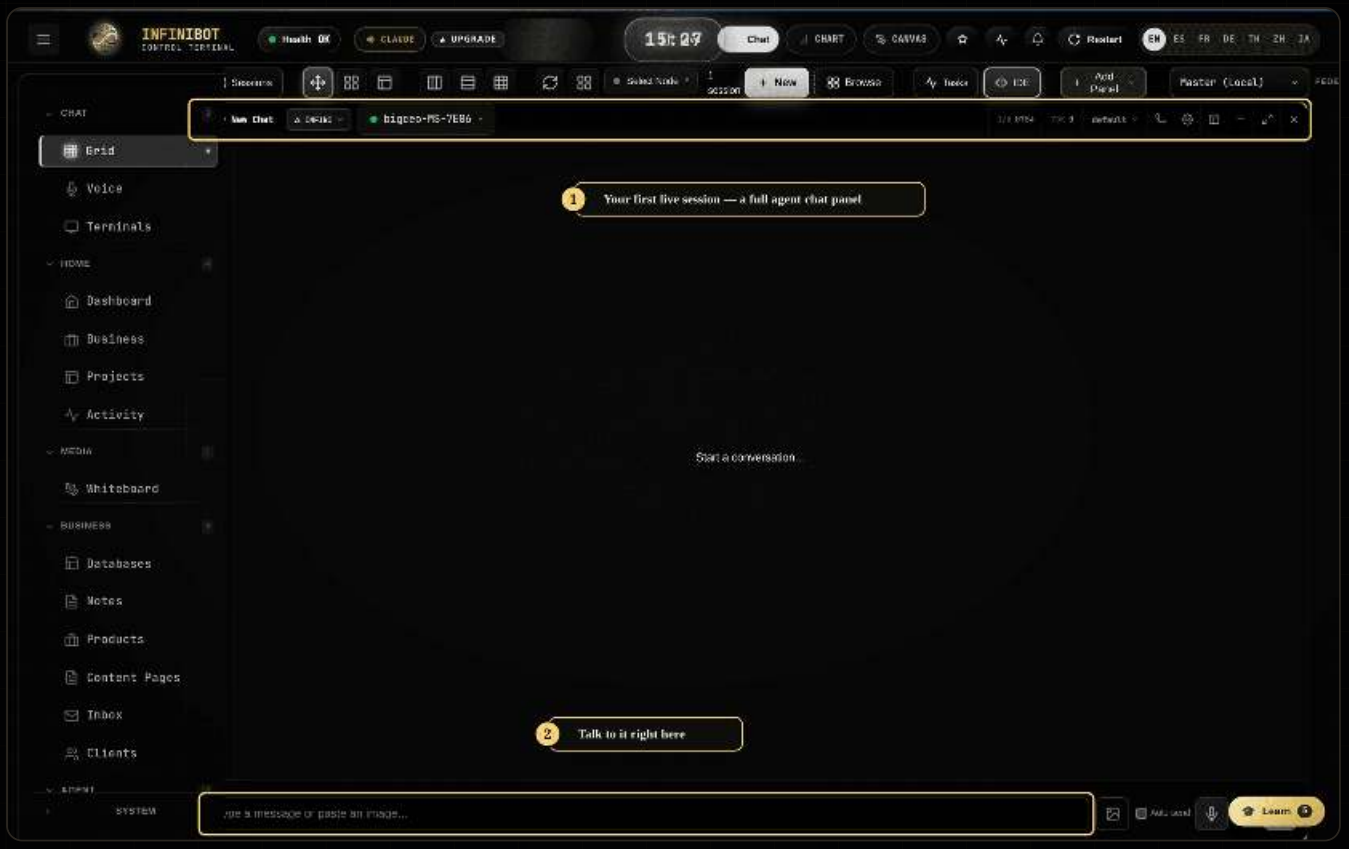


GRID STEP 1 — THE EMPTY MULTI-SESSION CANVAS; ARROW ① ON THE HERO, ② ON THE NEW BUTTON

② **Open the panel catalog.** Click **Add Panel**. The menu lists everything the grid can host: terminals (Shell, Claude Code, OpenCode, Gemini, Codex, Ollama), charts, a canvas, kanban boards, databases and live monitors.

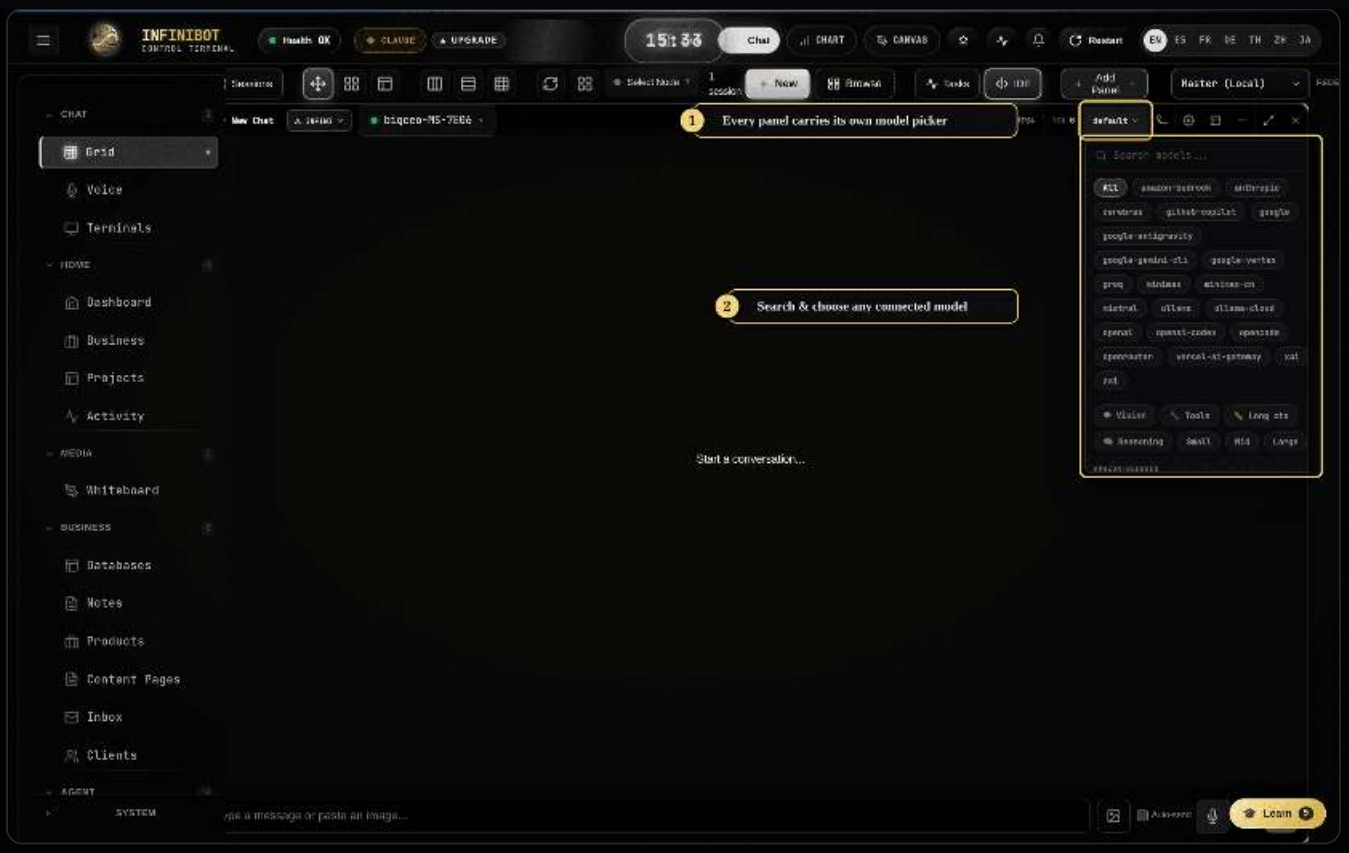
GRID STEP 2 — ADD PANEL CLICKED; THE OPEN CATALOG MENU IS CIRCLED

③ **Add a session.** Click **New**. A full agent chat panel drops in — header, model badge, and a composer at the bottom. Talk to it right there.



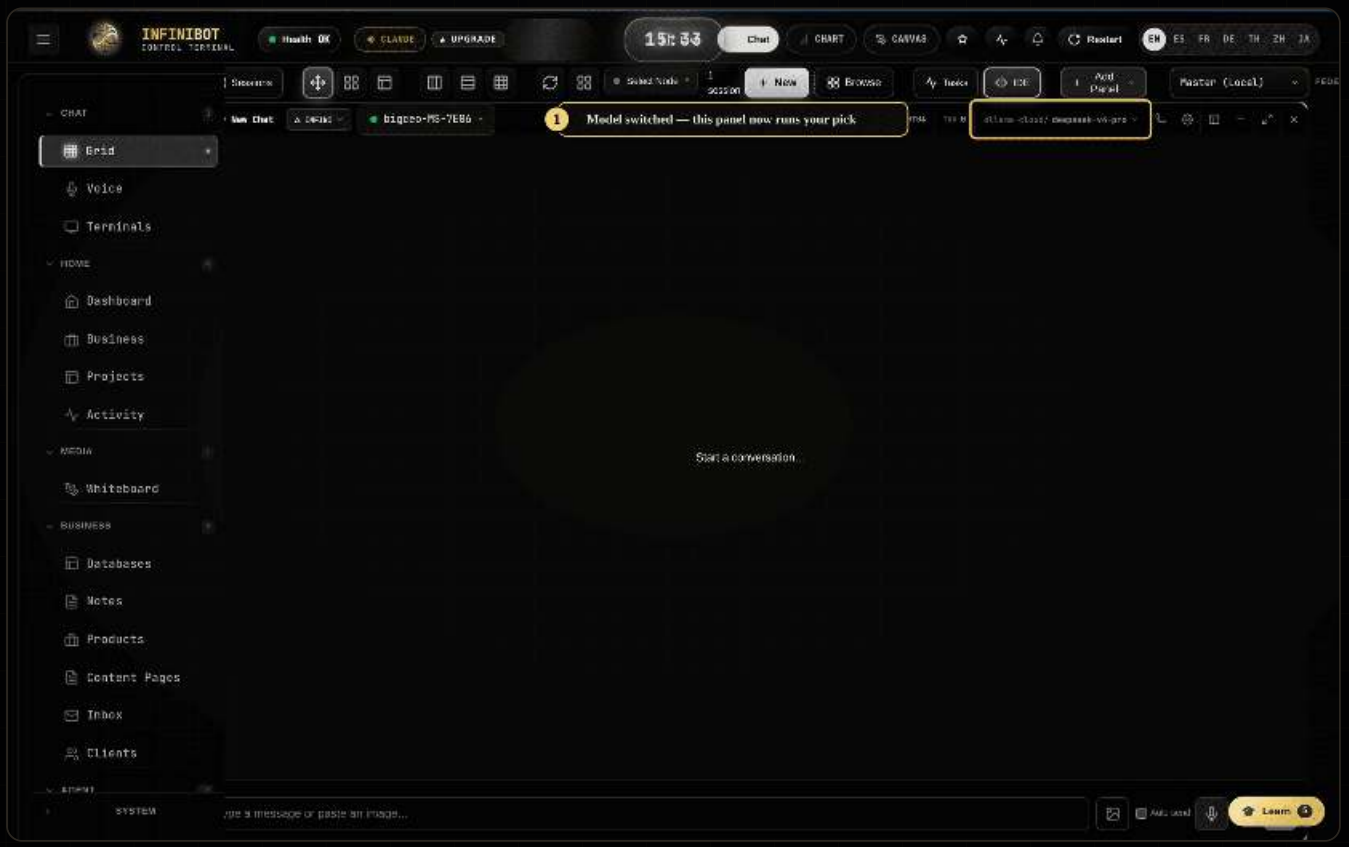
GRID STEP 3 — FIRST SESSION PANEL RENDERED; ARROW ① ON ITS HEADER, ② ON THE COMPOSER

④ **Open the panel's model picker.** Every panel carries its **own** model selector. Click it and the dropdown opens with a search box, provider filters (Anthropic, OpenAI, Google, Groq, Ollama, …) and capability/tier chips.



GRID STEP 4 — THE PANEL MODEL PICKER OPEN WITH THE FULL MODEL LIST CIRCLED

⑤ **Switch the model.** Pick one (here, the connected `deepseek-v4-pro`). The panel's badge updates — that panel now runs the model you chose, independent of every other panel.



GRID STEP 5 — MODEL SWITCHED; THE PANEL'S MODEL BADGE IS CIRCLED

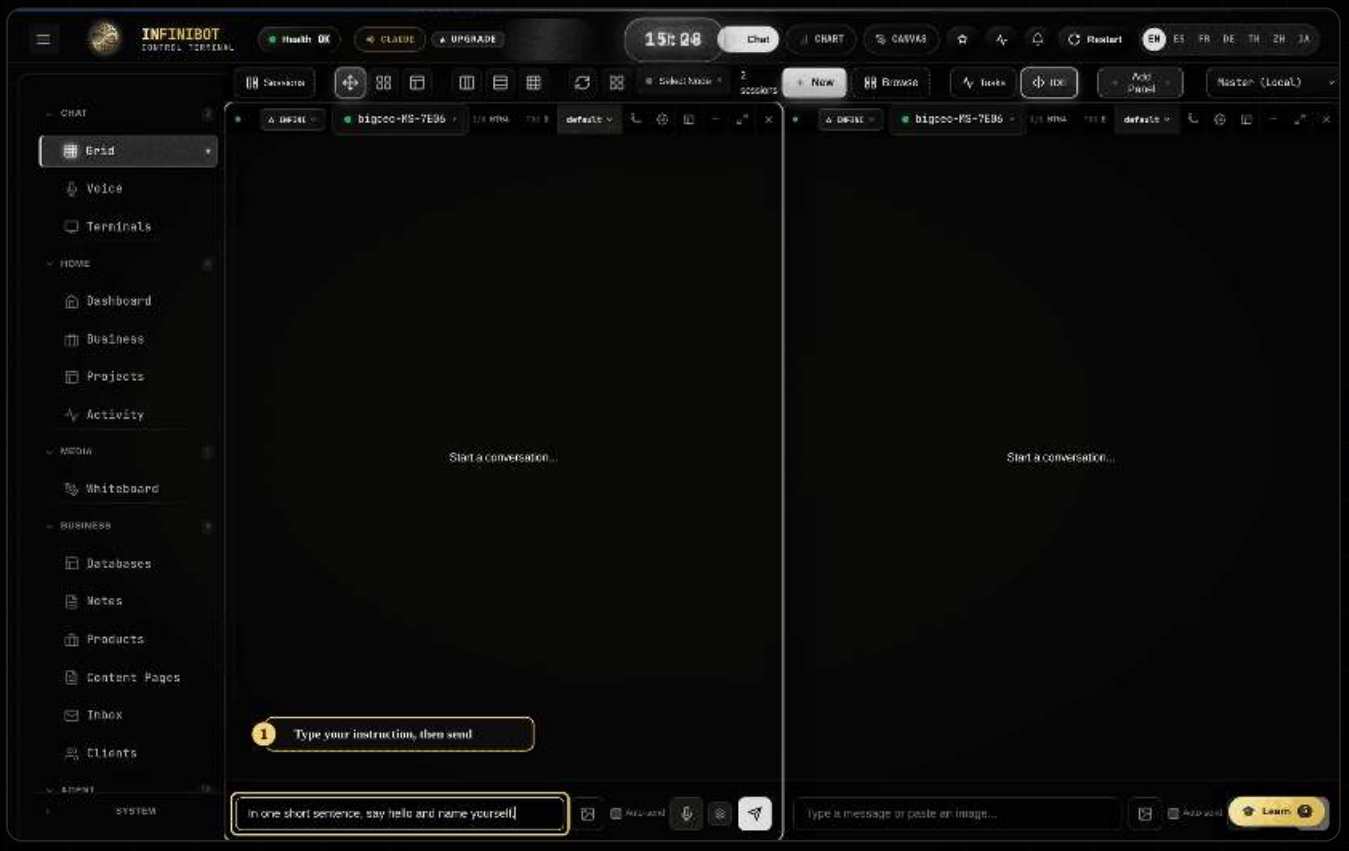
⑥ **Add a second session.** Click **New** again. Now two agents sit side by side, each driveable, each with its own context.

GRID STEP 6 — A SECOND SESSION PANEL ADDED; ARROW ① ON THE NEW PANEL

⑦ **Focus a panel.** Click any panel to focus it — it gets the gold active border, and keyboard input + voice route to it.

GRID STEP 7 — THE FOCUSED PANEL WITH ITS GOLD ACTIVE BORDER CIRCLED

⑧ **Type an instruction.** With a panel focused, type into its composer. Your text appears in that panel's input, ready to send.

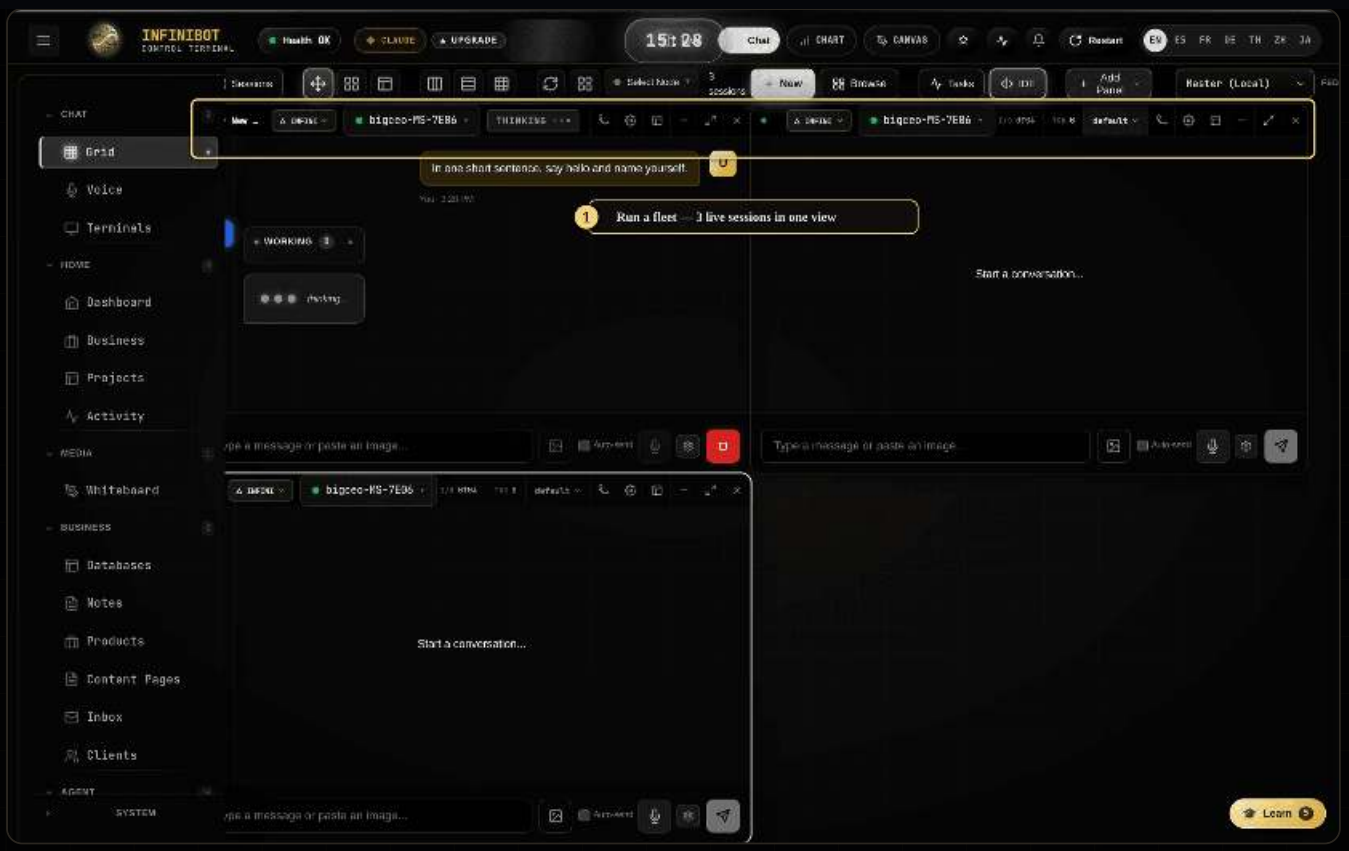


GRID STEP 8 — A PROMPT TYPED INTO THE FOCUSED PANEL'S INPUT, CIRCLED

⑨ **Send — the agent responds in-panel.** Hit send and the reply streams live, right inside the tile. No tab-switch, no lost context.

GRID STEP 9 — THE AGENT'S LIVE RESPONSE INSIDE THE PANEL, CIRCLED

⑩ **Run a fleet.** Add a third (or sixth, or ninth) panel and you're supervising a whole squad on one screen.



GRID STEP 10 — THREE LIVE SESSIONS TILED IN ONE VIEW

Time saved: Supervising 4 agents in 4 separate terminal windows means constant alt-tabbing and lost context. One grid = **~5–10 minutes of context-switching per hour reclaimed**, and you catch a derailing agent in seconds instead of after it has burned 10 minutes of tokens.

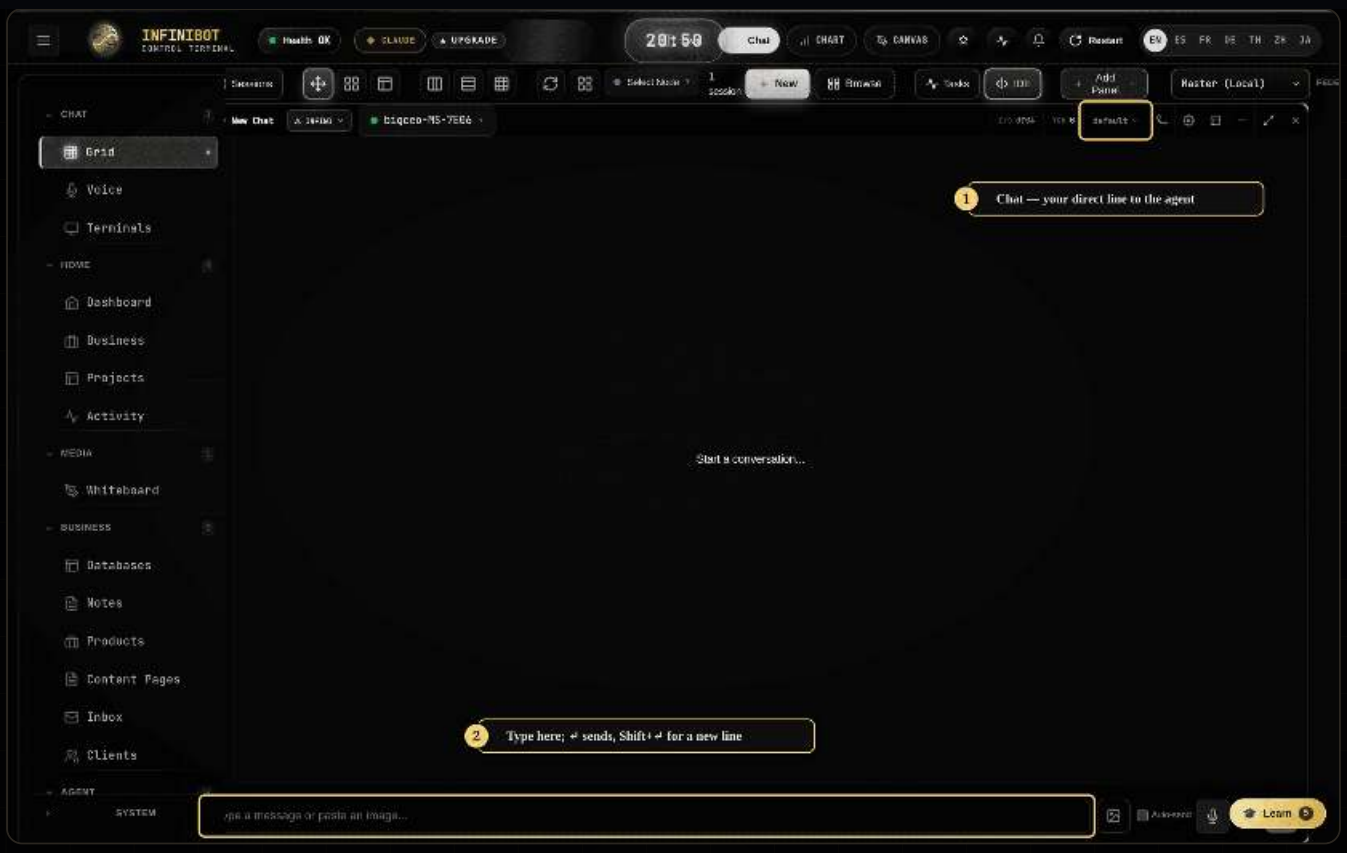
---

2 · Chat (`/chat`) — talk to an agent, click by click

Goal: Drive a single agent in a focused conversation — pick its model, send work, watch it answer, fire slash commands, and spin up more agents as tabs.

`/chat` is InfiniBot's **terminal chat**: a per-thread model picker, a working directory + project context bar, tool-call cards, slash-command autocomplete, and a tabbed thread strip where **each tab is its own agent**. Here is the full flow.

① **Open chat.** The composer sits at the bottom (↵ sends, Shift+↵ for a new line); the model picker is in the thread header.



CHAT STEP 1 — CHAT OPENED; ARROW ① ON THE MODEL PICKER, ② ON THE COMPOSER

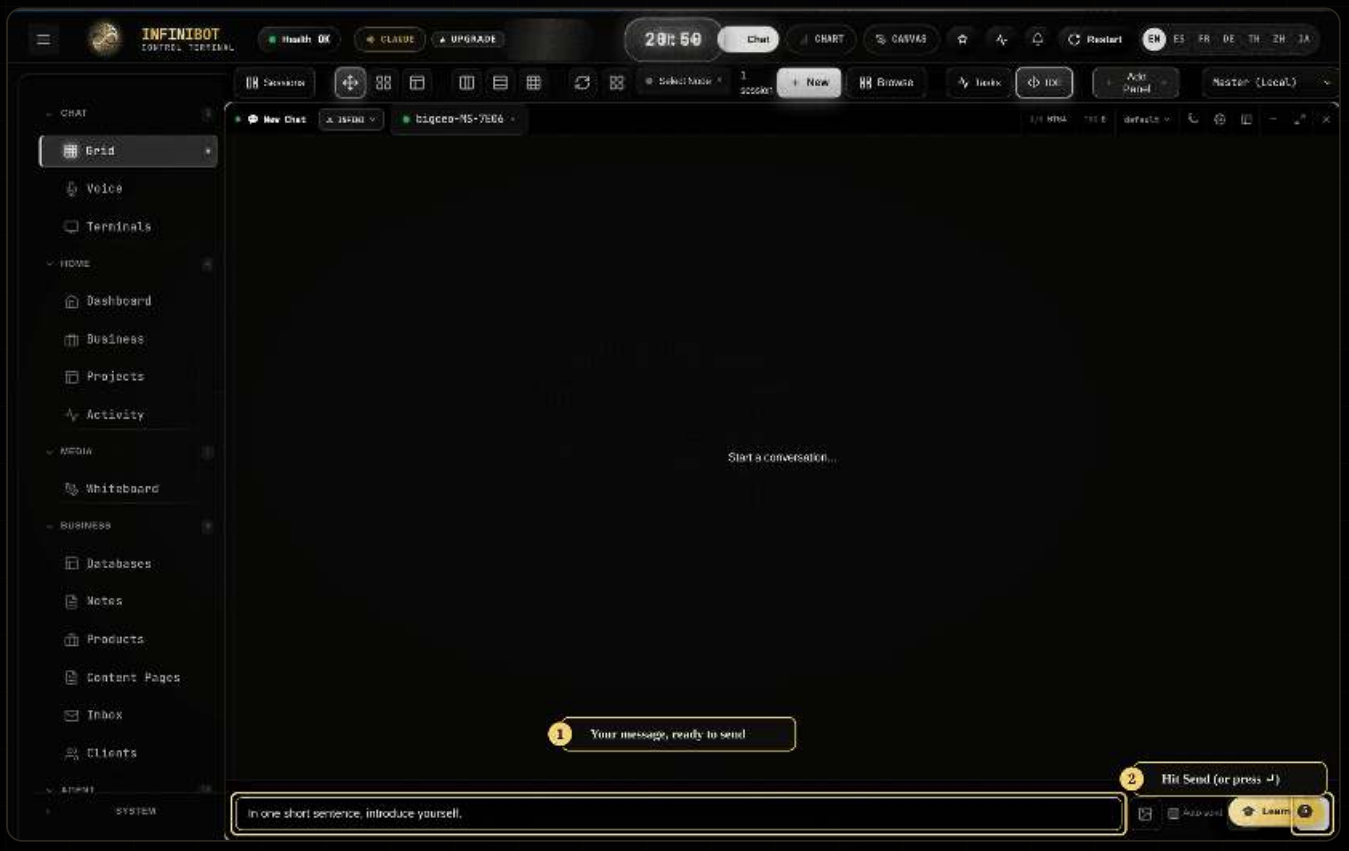
② **Open the model picker.** Click the model name in the header. The dropdown opens with a search box, provider filters and the full list of connected models.

CHAT STEP 2 — THE MODEL PICKER OPEN WITH THE MODEL LIST CIRCLED

③ **Pick a model.** Choose one and the header updates — this thread now runs the model you picked.

CHAT STEP 3 — MODEL SELECTED; THE HEADER MODEL BADGE CIRCLED

④ **Type your message.** Write the instruction in the composer. The gold **Send** button lights up.

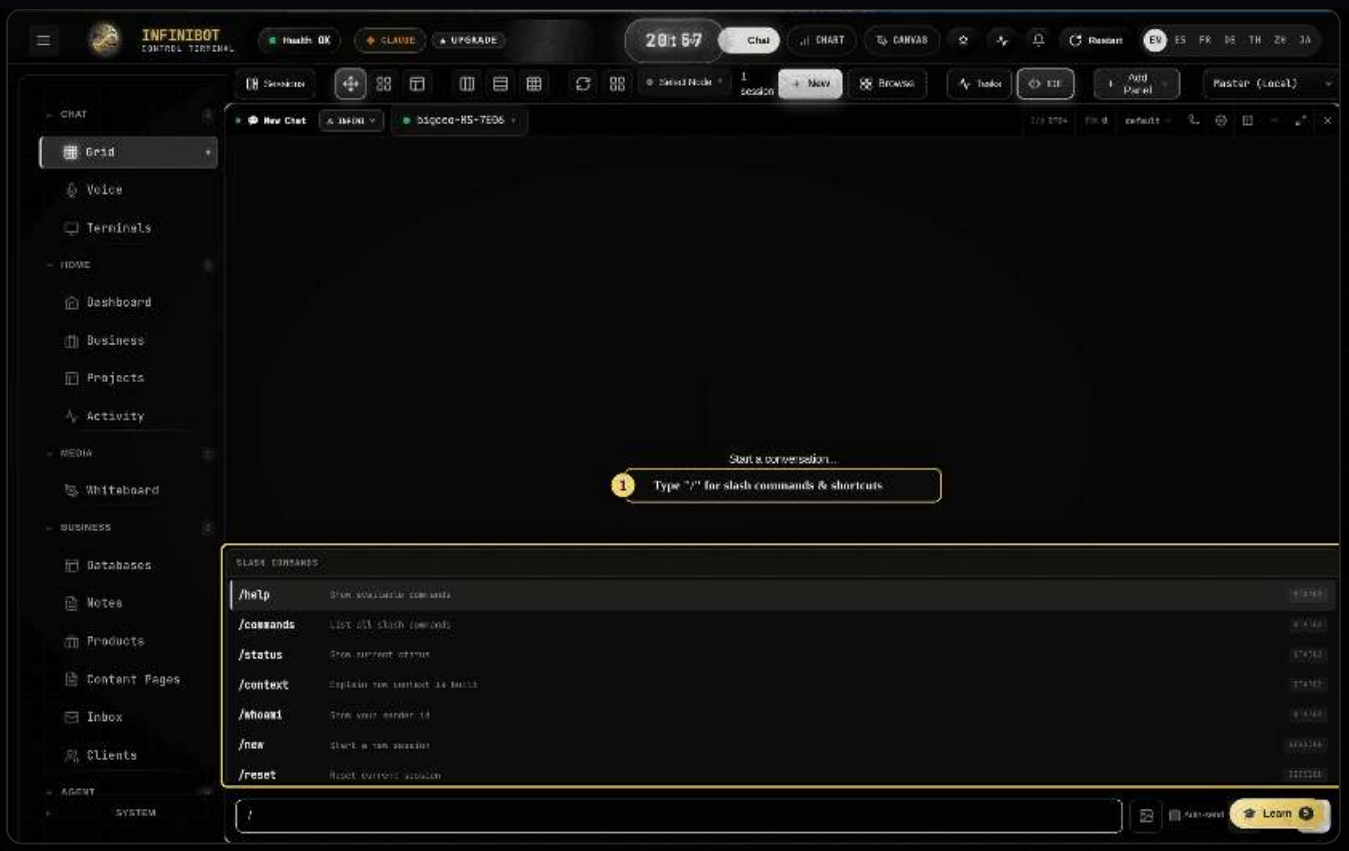


CHAT STEP 4 — A MESSAGE TYPED; ARROW ① ON THE INPUT, ② ON THE SEND BUTTON

⑤ **Send — the agent answers in real time.** The reply streams into the thread as an assistant message, with inline tool-call cards when the agent reaches for tools.

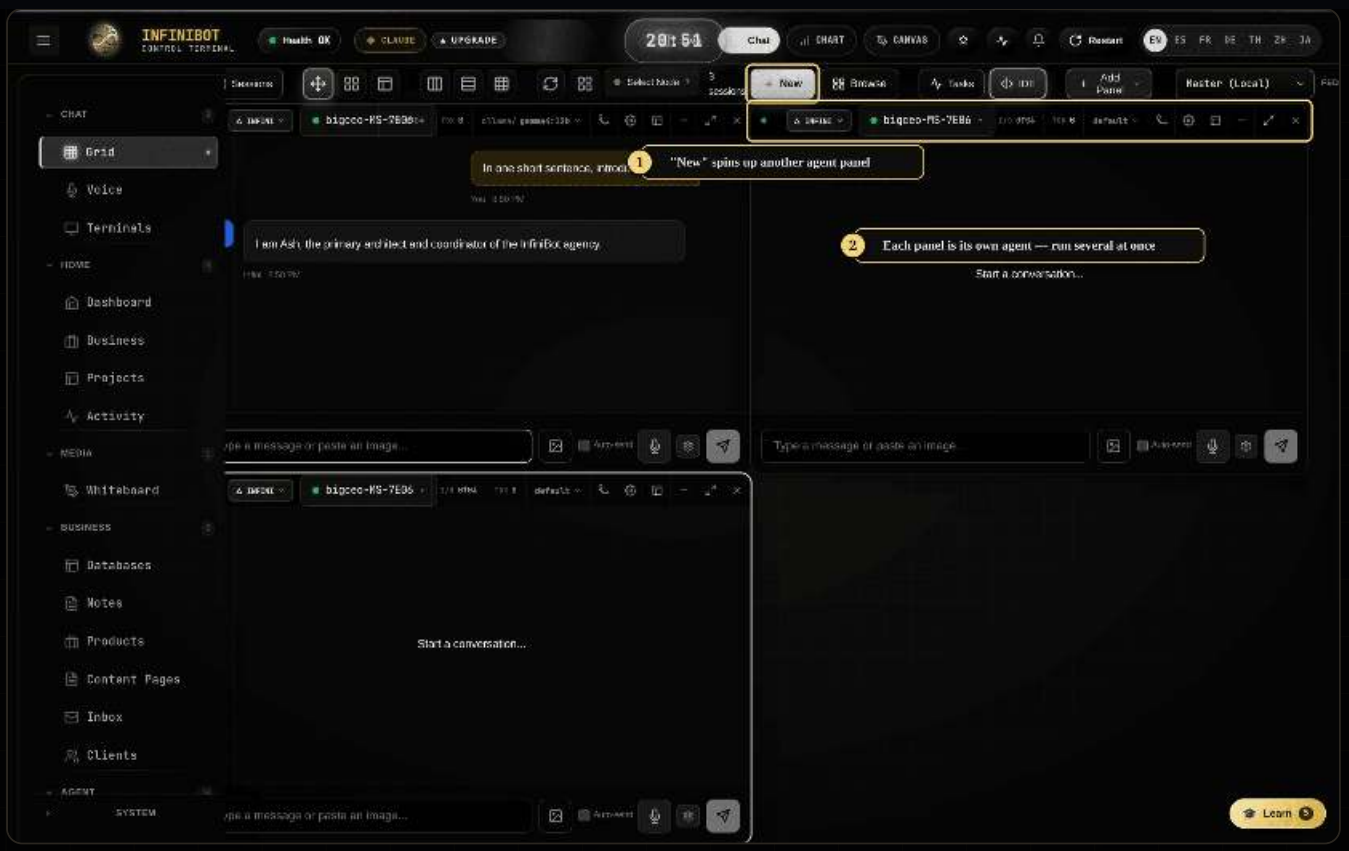
CHAT STEP 5 — THE AGENT'S REPLY LANDING IN THE THREAD, CIRCLED

⑥ **Slash commands.** Type `/` to open the slash-command menu — quick actions like `/project list`, `/project new`, `/project complete` without leaving the composer.



CHAT STEP 6 — THE SLASH-COMMAND MENU OPEN, CIRCLED

⑦ **Spawn more agents. New session** starts another agent thread, and the tab strip across the top lets you keep several agents going at once — each a fully independent session with its own context and transcript. Under the hood each spawn returns a `childSessionKey` + `runId` and is auto-tracked as a kanban task; pass a `projectHint` and the gateway fuzzy-matches an existing project instead of creating duplicates.



CHAT STEP 7 — ARROW ① ON NEW SESSION, ② ON THE AGENT TAB STRIP

Time saved: Four sequential 15-minute tasks = 60 minutes. Fan them across four chat tabs and they run concurrently ≈ 15–18 minutes wall-clock. **~70% faster** on any job that decomposes.

---

3 · Kanban /projects — your work queue + parent-project tree

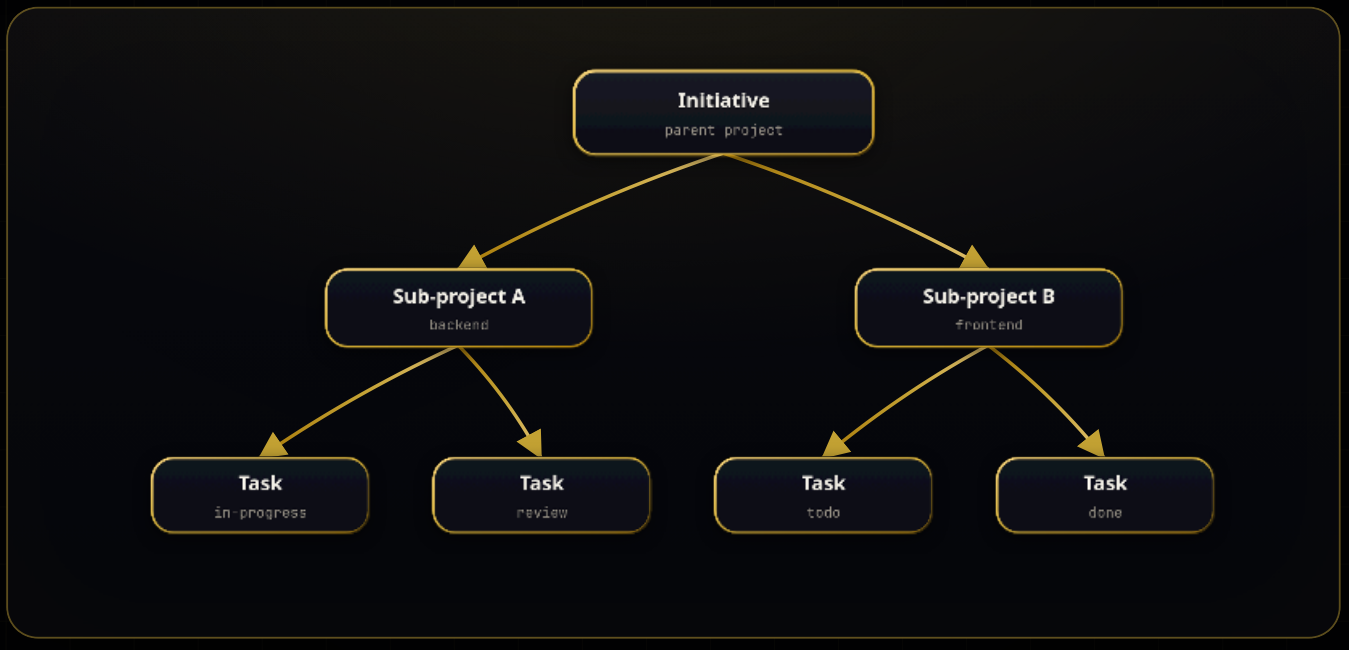
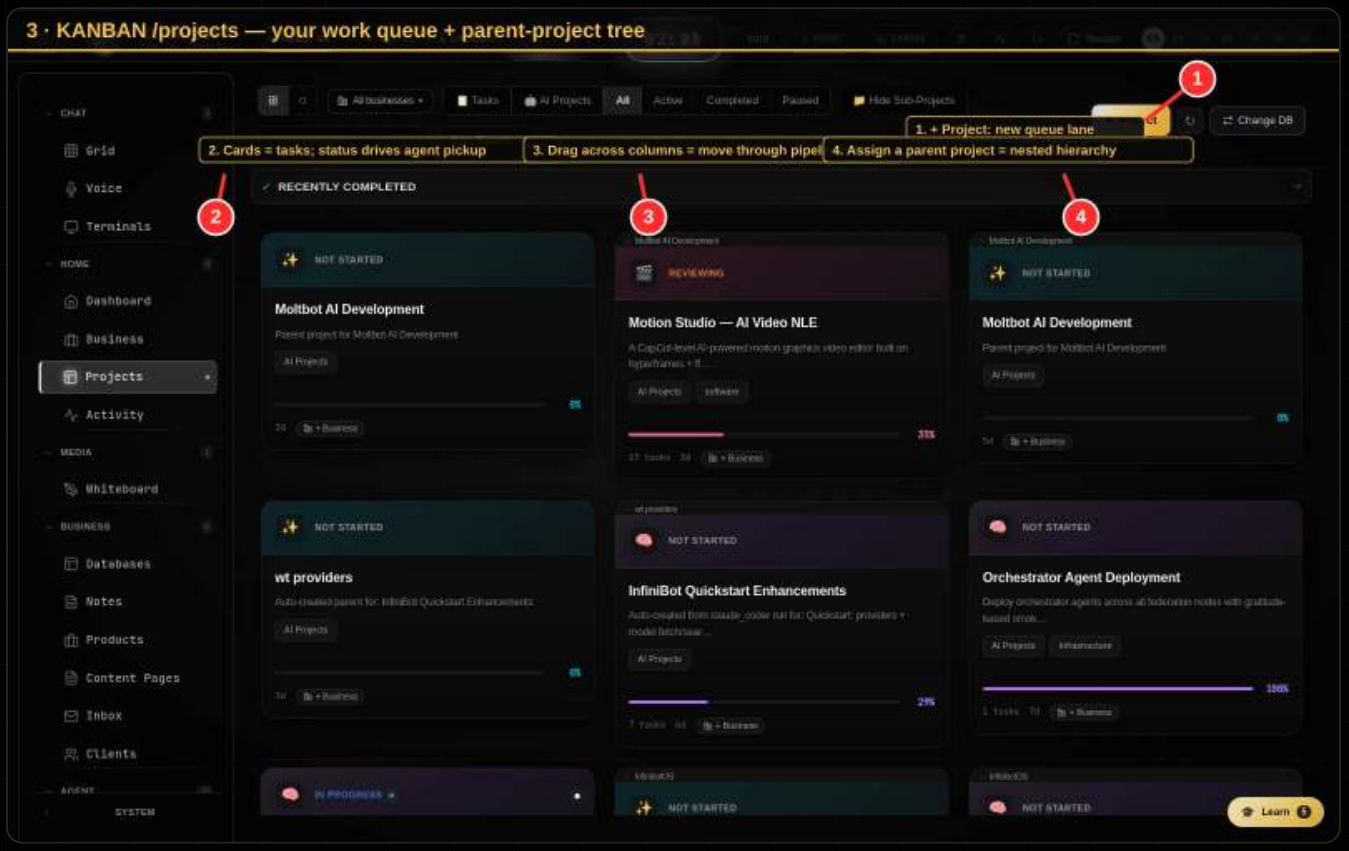


Figure · Project hierarchy — a parent initiative nests sub-projects, each a lane of task cards the agents pull from.

Goal: Treat the board as the agents' work queue, not just a status display.

Steps:

1. **+ Project** — create a queue lane for a body of work.
2. Cards are tasks; a card's **status** is what drives agent pickup and what the verification gate flips when a run finishes.
3. Drag cards across columns to move work through the pipeline (todo → in-progress → review → done).
4. **Assign a parent project** to nest related work — sub-projects roll up into a hierarchy so a big initiative stays one tree instead of a flat pile of cards.



PROJECTS QUEUE

> ⚠️ Changing a *project's* status cascades to **all** its child tasks. If you only mean to re-label the project, snapshot the done/in-progress tasks first.

Time saved: A shared board means you stop re-explaining "what's next" to each agent — the queue is the brief. **~15–20 minutes/day** of coordination overhead, and zero dropped tasks.

---

4 · Federation — dispatch to remote nodes via session keys

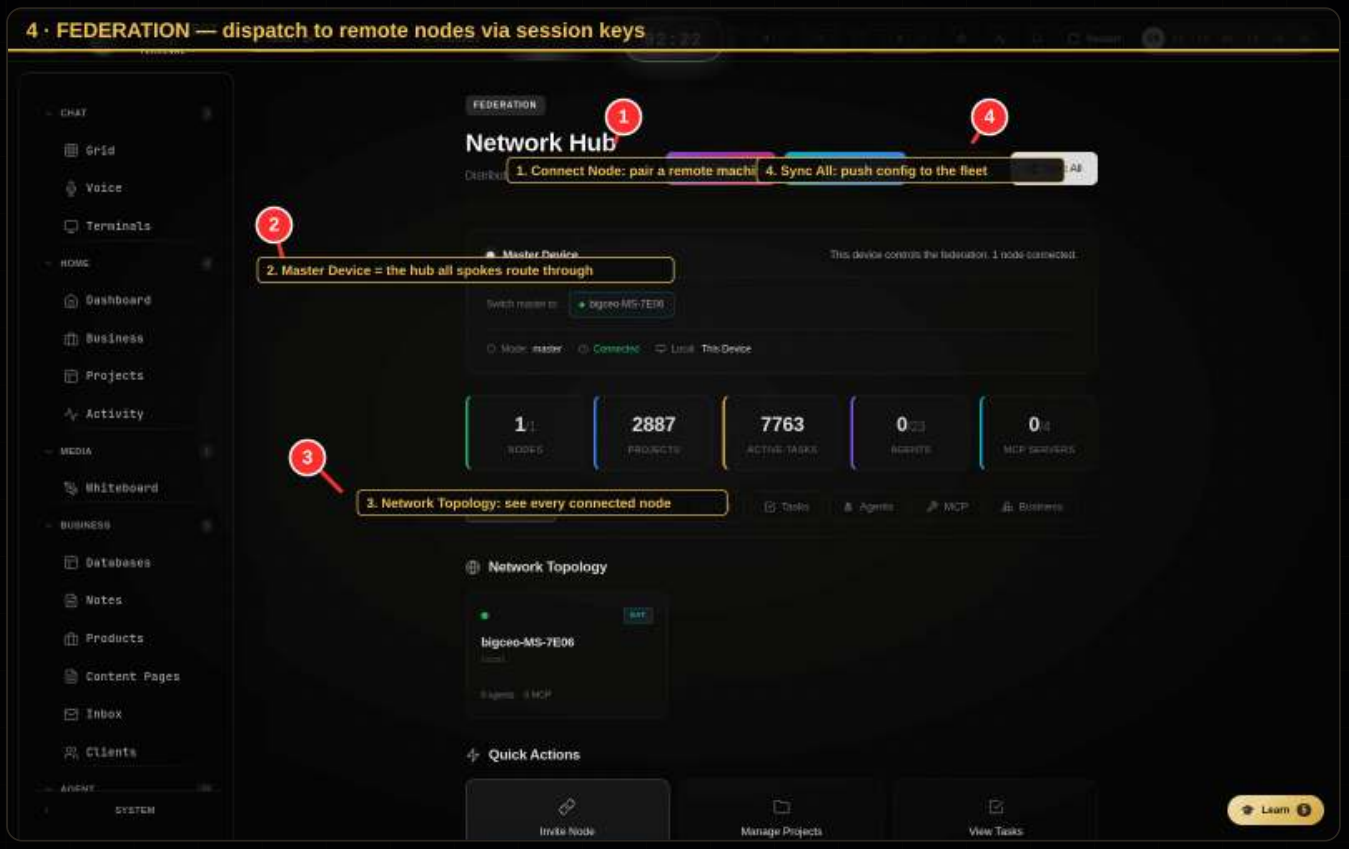
Goal: Use more than one machine. Send heavy jobs to a beefy box, keep light work local — all from one control terminal.

InfiniBot federation is **hub-and-spoke**: a Master node is the hub, every other machine is a spoke, and satellite → satellite traffic tunnels through the master.

Steps:

1. **Connect Node** — pair a remote machine (or run `satellite-setup.sh` on it).
2. The **Master Device** card is the hub every spoke routes through.
3. **Network Topology** lists every connected node and its health.
4. **Sync All** pushes config to the whole fleet at once.

Dispatch targets a node by **session key**. Local agents look like `agent:main:claude-coder:<runId>`; a sub-agent is `agent:coder:subagent:<uuid>`. Spawn with the node picked in the Grid's **Master (Local)** selector (§1b) or pass the node through `sessions_spawn`, and the call is forwarded over the hub.



FEDERATION NODES

Time saved: Offloading a 20-minute build to a spare workstation while you keep working locally is **pure parallelism** — the remote job is "free" wall-clock time. On a 3-machine fleet that's effectively **3× throughput** for parallelizable work.

---

5 · Workflows — one-shot AND kind:continuous background loops

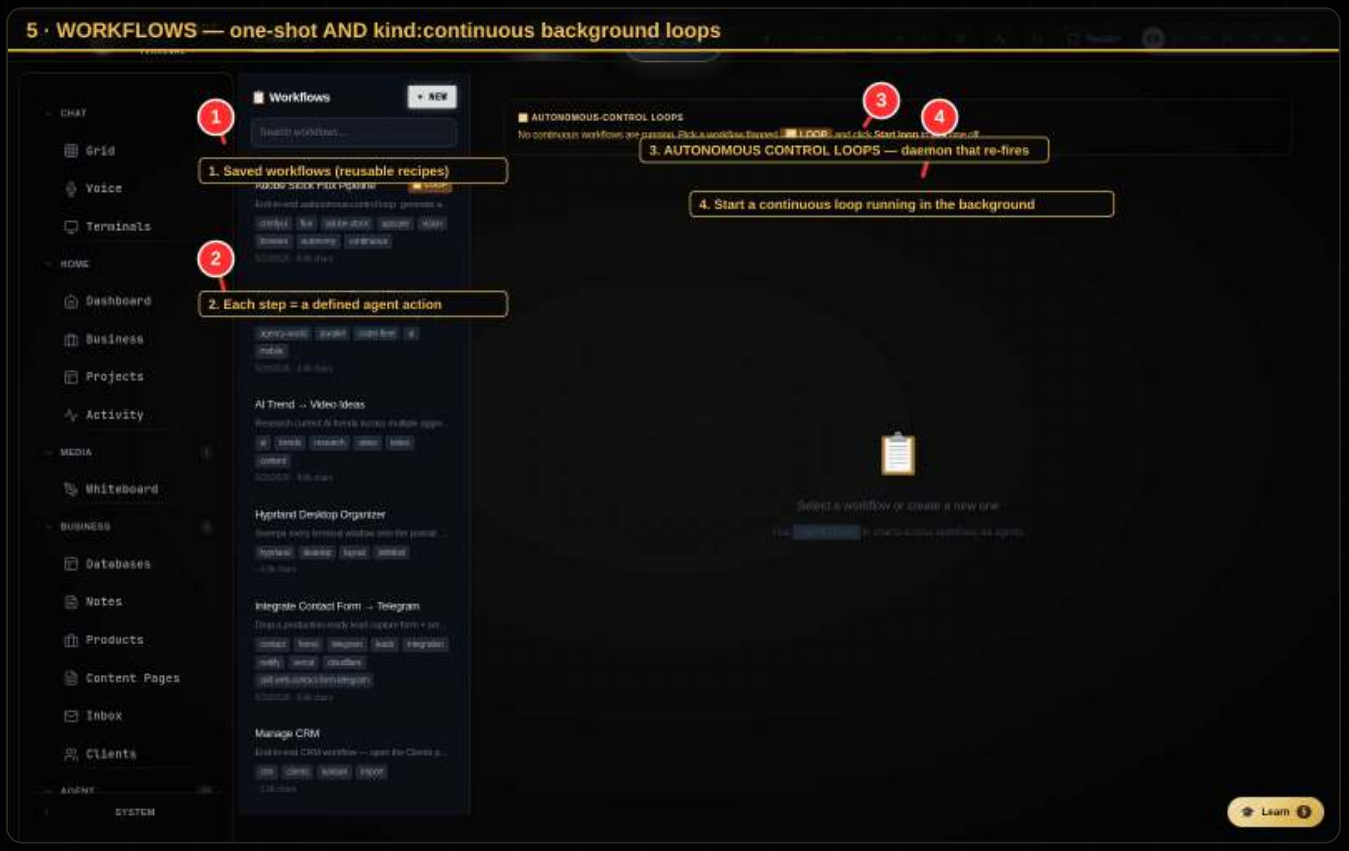
Goal: Stop re-typing multi-step processes. Save them as workflows; some run once, others **loop in the background forever**.

InfiniBot has **two kinds** of workflow:

- **One-shot** — a saved recipe of steps you `run` on demand.
- `kind: continuous` — added 2026-05-23 — a **daemon loop** that re-fires on its own (the *Autonomous Control Loops*). Use it for "keep doing X until I say stop": triage an inbox, watch a queue, re-run a health check.

Steps:

1. Browse **saved workflows** in the left list — each is a reusable recipe.
2. Each row expands to its **defined steps** (each step = an agent action).
3. The **Autonomous Control Loops** panel is where continuous workflows live — a daemon that re-fires until you stop it.
4. **Start** a continuous loop and it runs in the background while you do other work.



CONTINUOUS WORKFLOWS

Time saved: A 6-step process you run daily by hand is ~~8 minutes of clicking. As a one-shot workflow it's one click. As a continuous loop it's **zero clicks** — it just happens.~~ **40+ minutes/week** for any recurring chore.

---

6 · MCP tool stacking — one agent, many tool servers

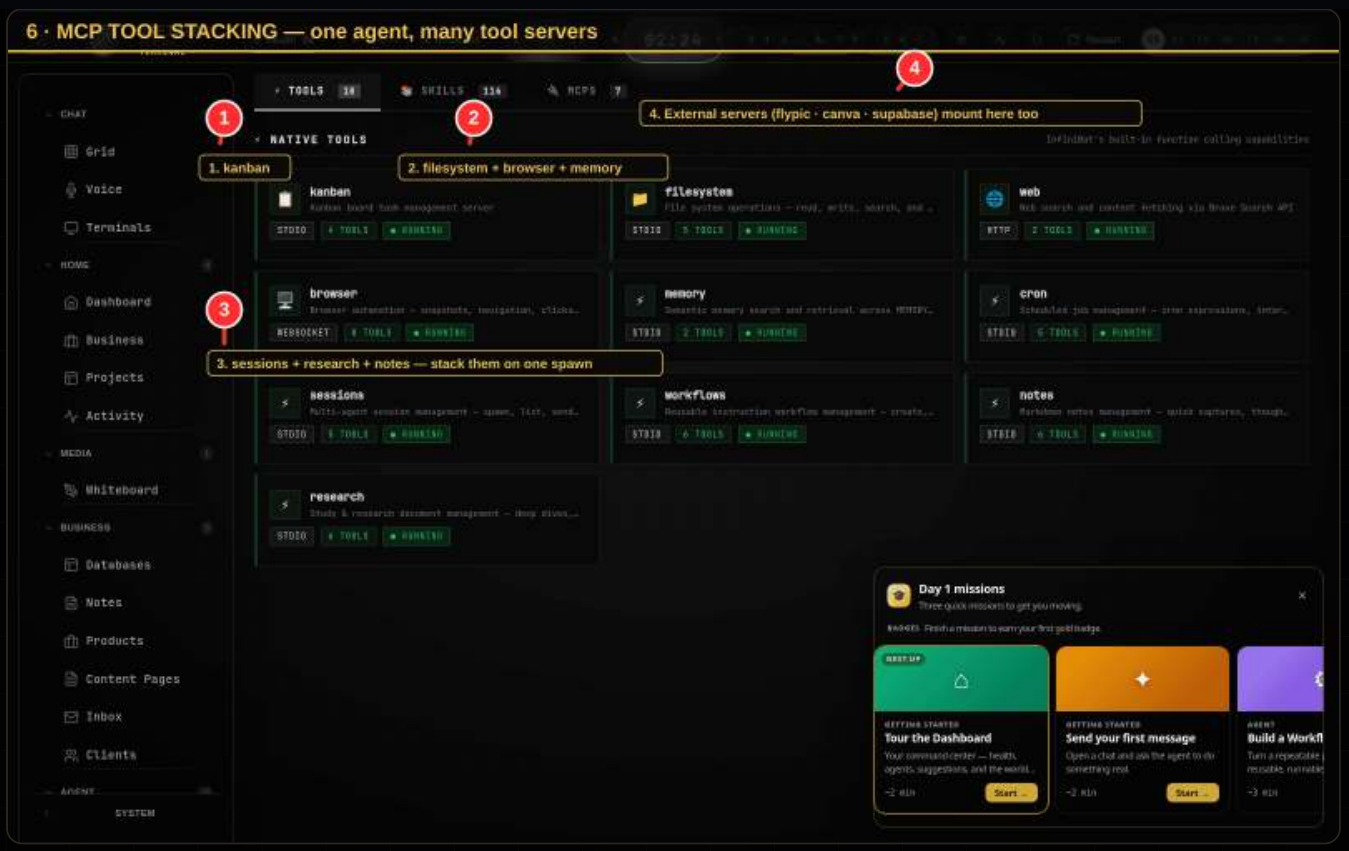
Figure · MCP tool stacking — one agent reaches a live stack of tool servers, doing research → design → store → publish in a single run.

Goal: Give a single agent a *stack* of capabilities so it can do an end-to-end job without you stitching tools together.

The `/mcp` registry shows every tool server the gateway exposes. The surface is **dynamic** — external servers are derived live from `agent.tools.list`, so what an agent can reach is whatever's connected, not a hardcoded list.

Steps:

1. **kanban** — the agent reads/writes the work queue itself.
2. **filesystem + browser + memory** — read code, drive a real browser, recall prior sessions.
3. **sessions + research + notes** — stack these on a single spawn via `enabledMCPs` so the agent can spawn helpers, pull research, and write notes in one run.
4. **External servers** — flypic, canva, supabase, excalidraw and friends mount in the same registry. Stack them and one agent can *research a product → generate the visual → write it to Supabase → drop a Canva export* without a human handoff.



MCP TOOL STACKING

Time saved: A "research → design → store → publish" chain done by hand across four apps is ~~30–45 minutes of copy-paste and context loss. One stacked agent does it in a single run. **~~80% per cross-tool task.**

---

7 · Knowledge graph — instant context retrieval

Goal: Hand an agent exactly the right context instead of letting it re-discover your project from zero every time.

The knowledge graph (`/knowledge-graph` , a d3-force canvas) is every project, task, note, agent, session and memory as connected nodes. Dense hubs are your highest-recall context.

Steps:

1. The **dense central hub** is the most-connected memory — the context most worth feeding an agent.
2. **Search / filter** (the category chips: Projects · Tasks · Notes · Agents · Sessions · Databases · Runs · Skills · Federation · Users) to the nodes that feed the task at hand.
3. **Edges** are the relationships agents traverse to pull related context — follow them to find what's connected to what.

KNOWLEDGE GRAPH

Time saved: Re-briefing an agent on project background is ~~5–10 minutes of prose per spawn. Pointing it at the graph means it retrieves context itself.~~ **5 minutes saved per agent, every spawn** — and far fewer "the agent didn't know about X" reworks.

---

8 · Discipline auto-log + proactive alerts — keep the loop honest

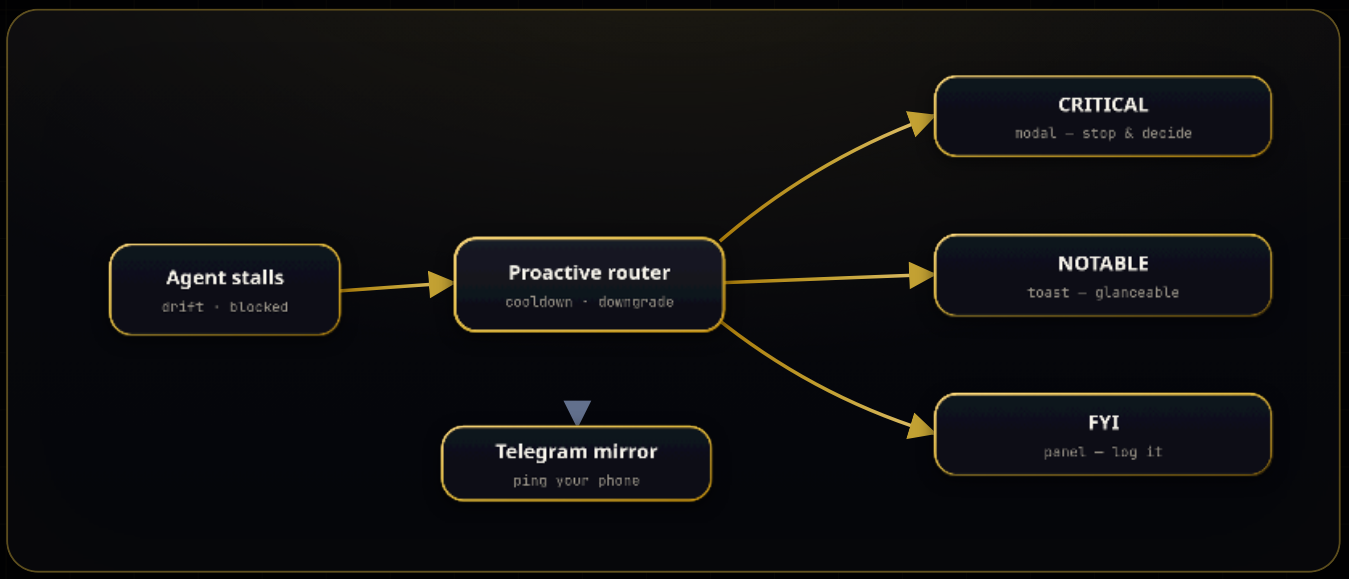
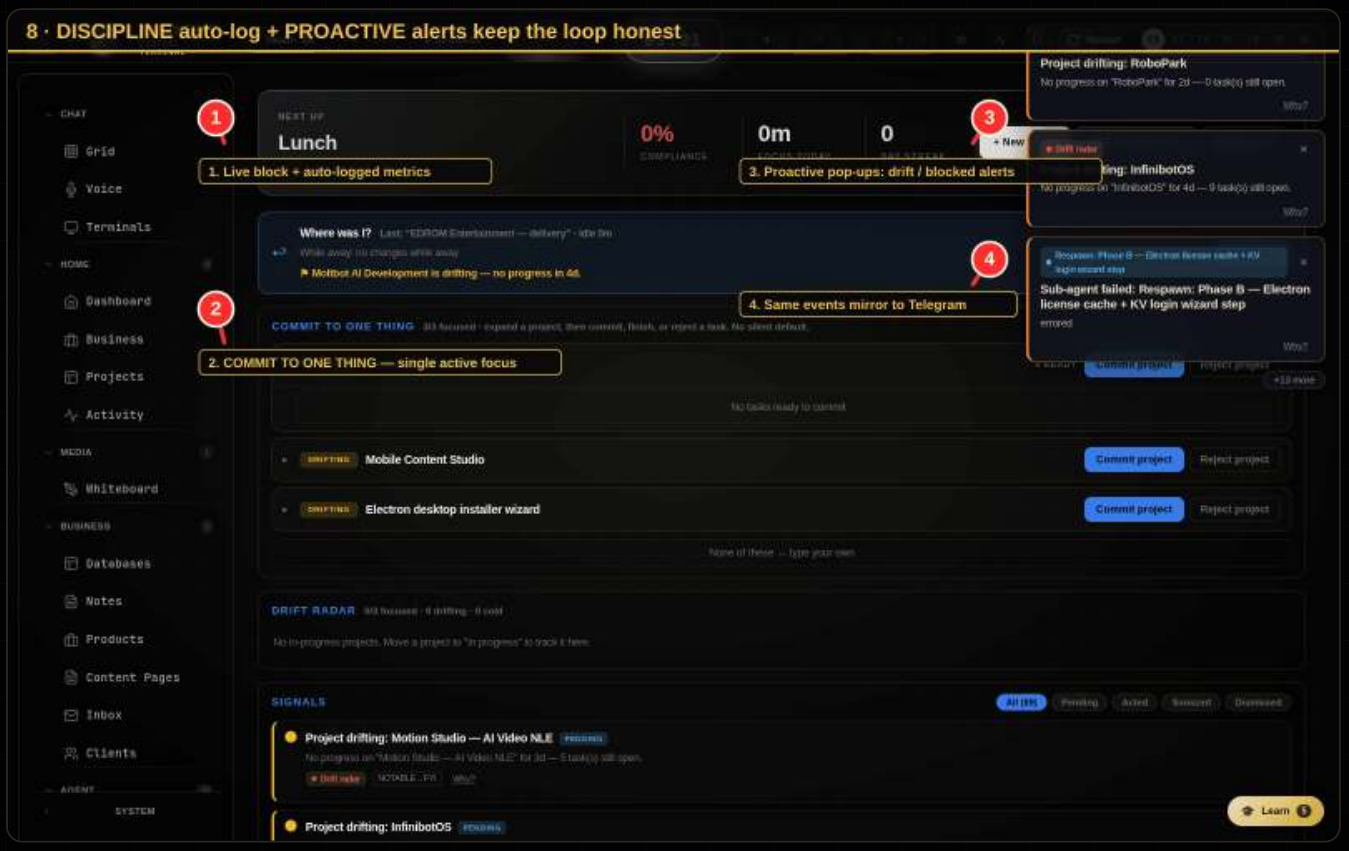


Figure · Proactive tiers — a stalled agent routes through the proactive router into severity tiers and mirrors to Telegram.

Goal: A fleet that runs itself still needs a human in the loop. The discipline system + proactive pop-ups make sure drift surfaces *to you* instead of rotting silently.

Steps:

1. The **live block + auto-logged metrics** track what you (and the agents) actually did — logged automatically, not by hand.
2. **Commit to one thing** enforces a single active focus so you don't fan yourself too thin while the fleet works.
3. **Proactive pop-ups** raise drift / blocked / needs-decision alerts in tiers (CRITICAL modal · NOTABLE toast · FYI panel) the moment an agent stalls.
4. The same events **mirror to Telegram** (server-side bridge) so you get pinged on your phone when something needs you — the loop stays honest even away from the desk.



DISCIPLINE + PROACTIVE

Time saved: Catching a blocked agent in 30 seconds via a push instead of discovering it 40 minutes later on a manual check. **~30–40 minutes of wasted agent runtime avoided per incident**, plus you stop babysitting screens.

---

9 · Focus modal — in-UI file diff / revert / accept + check-in gates

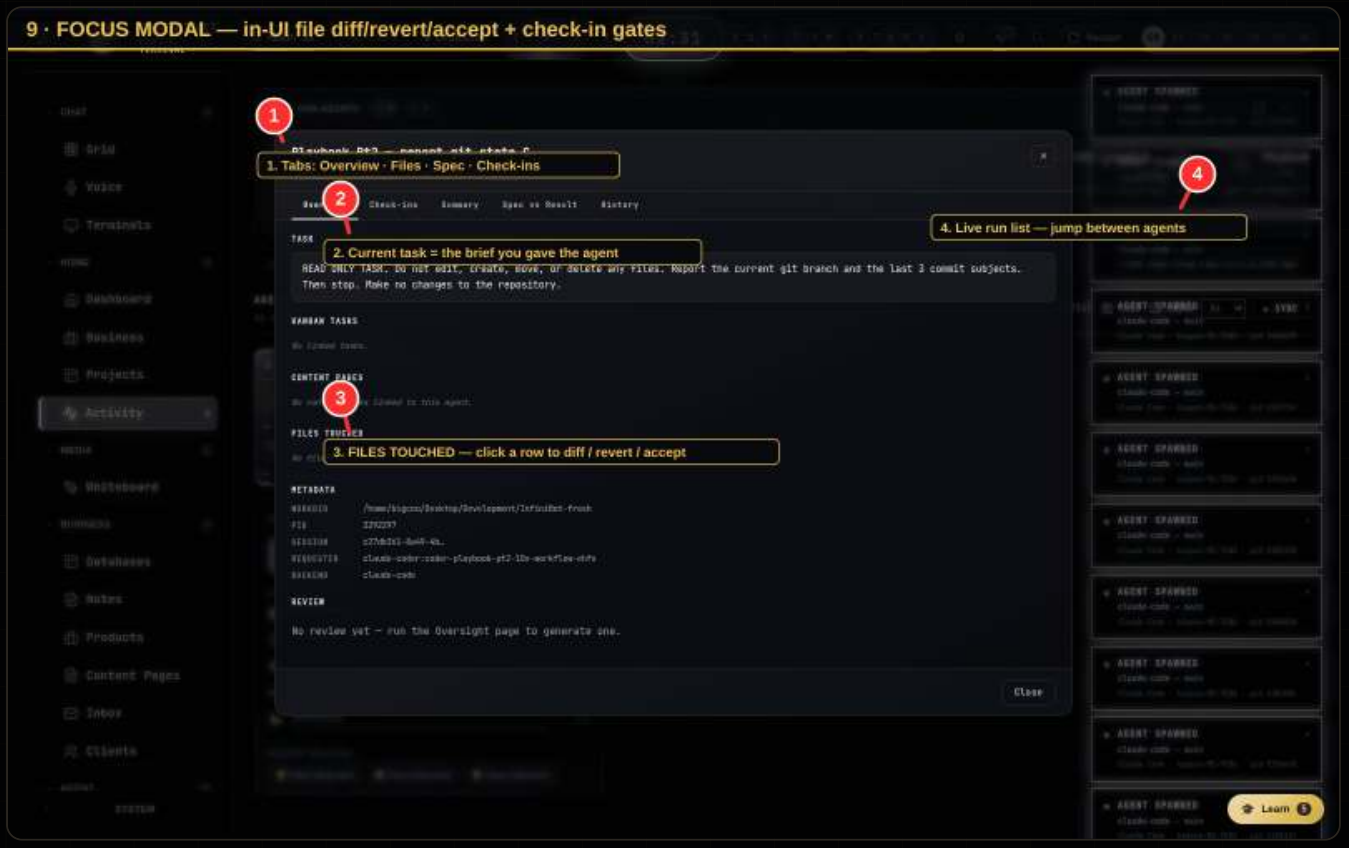
Goal: Review and govern an agent's work without leaving the browser — and, when it matters, make the agent *pause for your approval* before it drifts.

Open any run's focus modal (dispatch `infinibot:subagent-open-detail`, or click a run on `/activity`). Everything about that agent is here.

Steps:

1. **Tabs** — Overview · Files · Spec · Check-ins — the full picture of one run.
2. The **current task** is the exact brief you handed the agent (your spec of record).
3. **Files touched** — click any row to **diff / revert / accept** the change right in the drawer (backed by `files.diff` / `files.revert` / `files.accept`). No terminal.
4. The **live run list** lets you jump between agents without closing the modal.

For high-stakes work, spawn with `requireCheckpointApproval: true`. Every `report_checkpoint` the agent emits then **pauses the run** until you click **Approve / Redirect / Stop** in the Check-ins tab — a hard gate against drift.



FOCUS MODAL

Time saved: Reviewing a diff, reverting a bad hunk, and accepting the rest — all in-browser — replaces a `git diff` / stash / cherry-pick dance. **~5–10 minutes per review**, and the check-in gate prevents the much costlier "agent went off the rails for 20 minutes" failure mode.

---

10 · World map — see throughput as live operations

Goal: Feel the throughput. The world map turns the abstract "12 agents working" into a living operations view you can read at a glance.

Steps:

1. **Live operations** renders every agent as a unit on an isometric map.
2. Agents **route to tool sectors** as they work — you can literally see who's building, who's researching, who's idle.
3. The **timeline scrubber** replays the day's activity — scrub back to see what happened while you were away.
4. **Drift alerts** surface right beside the map so problems don't hide behind the pretty visualization.

WORLD MAP

Time saved: This is less about minutes and more about **situational awareness** — one glance answers "is the fleet busy and healthy?" that would otherwise take a scan of five different views. ~2 minutes per status check, many times a day.

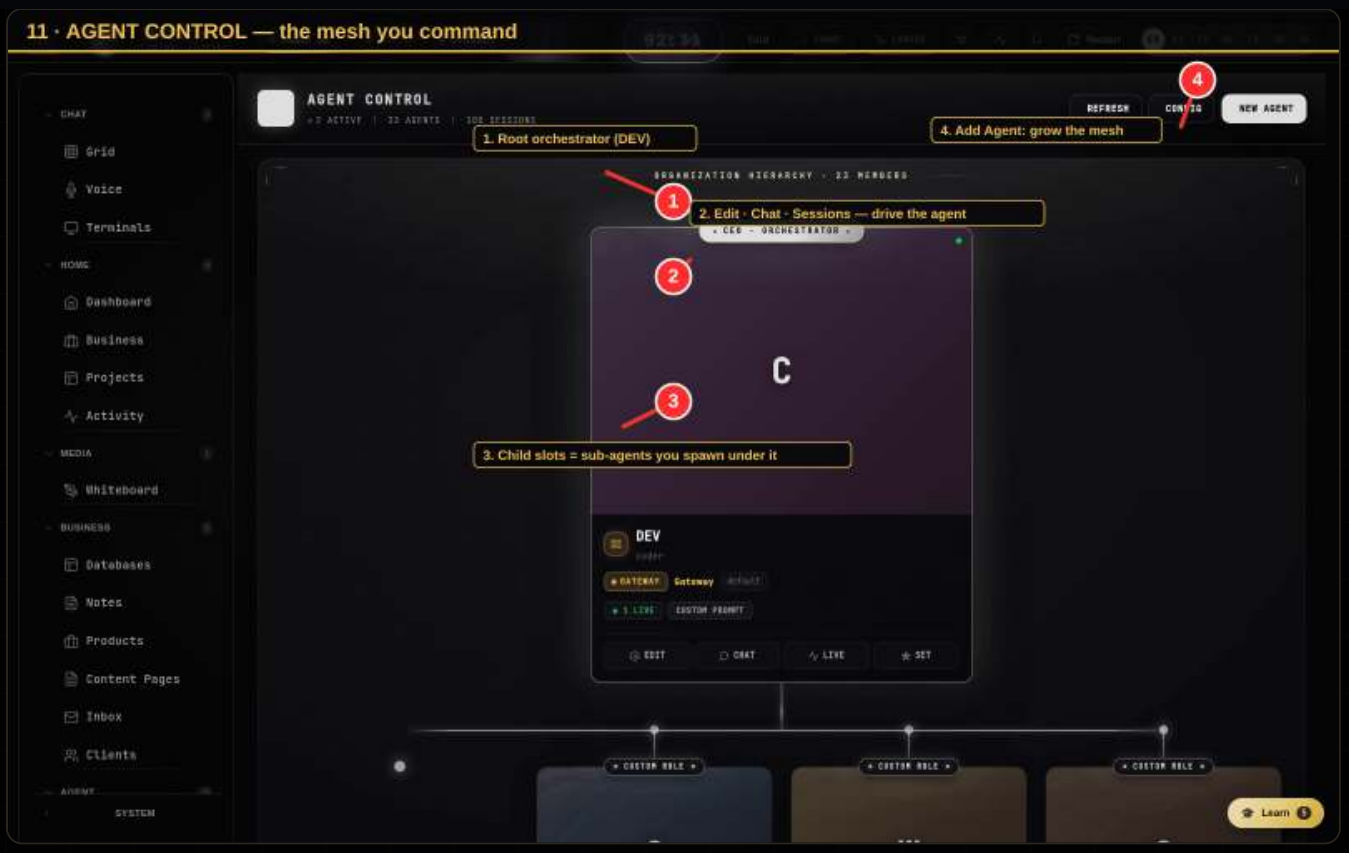
---

11 · Agent Control — the mesh you command

Goal: Manage the whole agent mesh as a structure, not a list — who reports to whom, who can spawn whom.

Steps:

1. The **root orchestrator** (DEV) sits at the top of the mesh.
2. **Edit · Chat · Sessions** drive that agent directly from its node.
3. **Child slots** are the sub-agents you spawn underneath it — the mesh grows downward as you delegate.
4. **Add Agent** grows the mesh with a new identity.



AGENT CONTROL

Time saved: A clear hierarchy means you delegate to *one* orchestrator and it fans work to its children, instead of you micromanaging every leaf. **Removes a layer of manual dispatch** on any multi-stage job.

---

12 · Credits — prove the throughput (real spend & tokens)

Goal: Quantify the 10x. The credits page is the receipt: real spend and real tokens, per backend.

Steps:

1. **Per-backend spend** — Claude / Codex / OpenCode / Gemini side by side.
2. The **plan usage ring** shows how much runway is left before you hit a cap.
3. **Tokens by** `agent_backend` (the donut) shows *where* the throughput went — which backend did the heavy lifting.
4. **Total spend vs. budget** tracks the whole fleet against your limit.

CREDITS THROUGHPUT

Time saved: Knowing your real cost-per-outcome lets you route the next job to the cheapest capable backend instead of guessing. Over a month that's the difference between **burning budget and compounding it.**

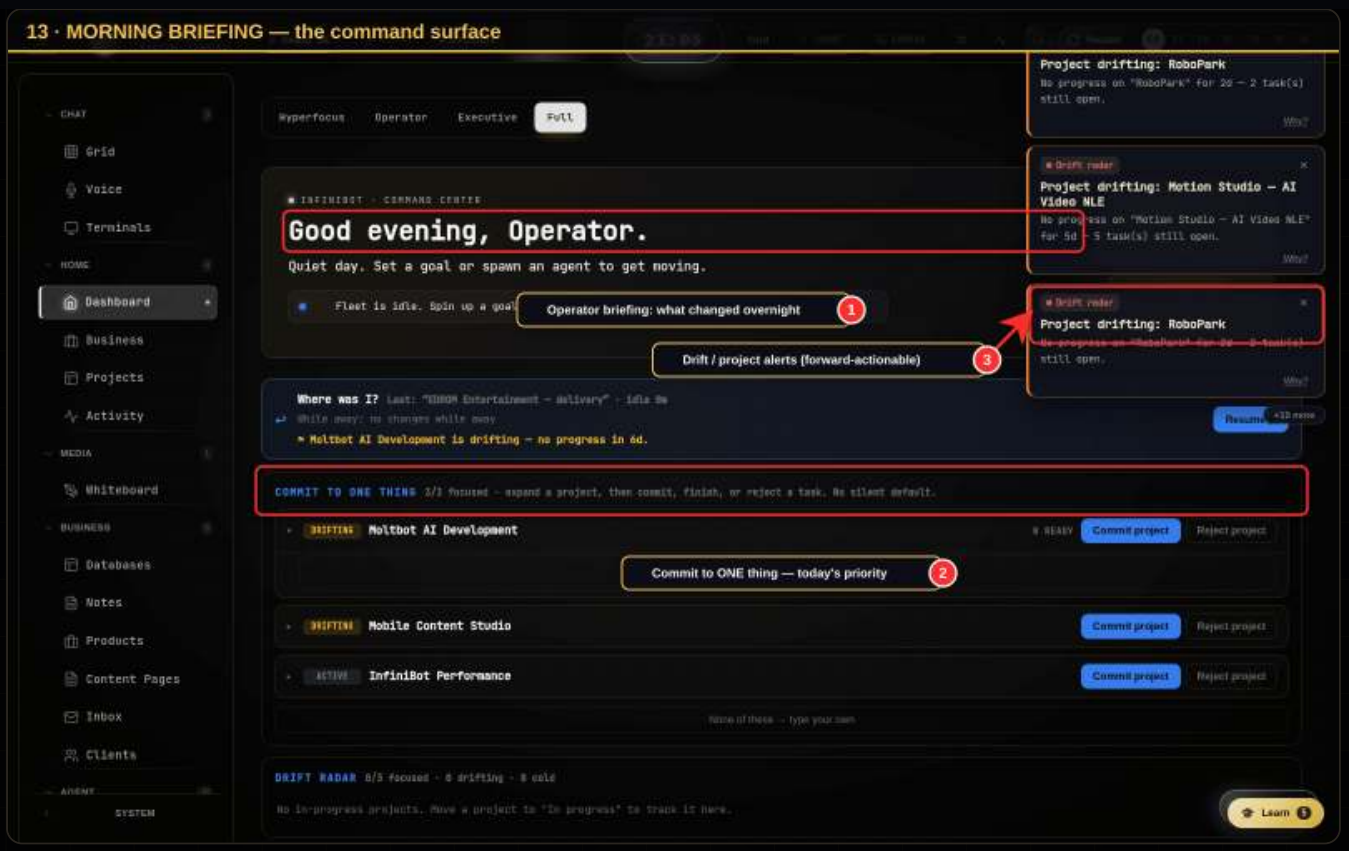
---

13 · Morning briefing — the command surface

Goal: Start every day already oriented. The dashboard briefing is the cockpit.

Steps:

1. The **operator briefing** summarizes what changed overnight — agent runs, completions, new alerts.
2. **Commit-to-one-thing** surfaces your single highest-priority focus.
3. **Drift / project alerts** are forward-actionable — each card has a button that spawns a fix, navigates, or dismisses, so you act from the briefing itself.



MORNING BRIEFING

Time saved: Replacing a 10-minute "what's the state of everything?" catch-up with a 60-second briefing. **~10 minutes every morning**, and you start the day acting instead of orienting.

Putting it together — a day at 10x

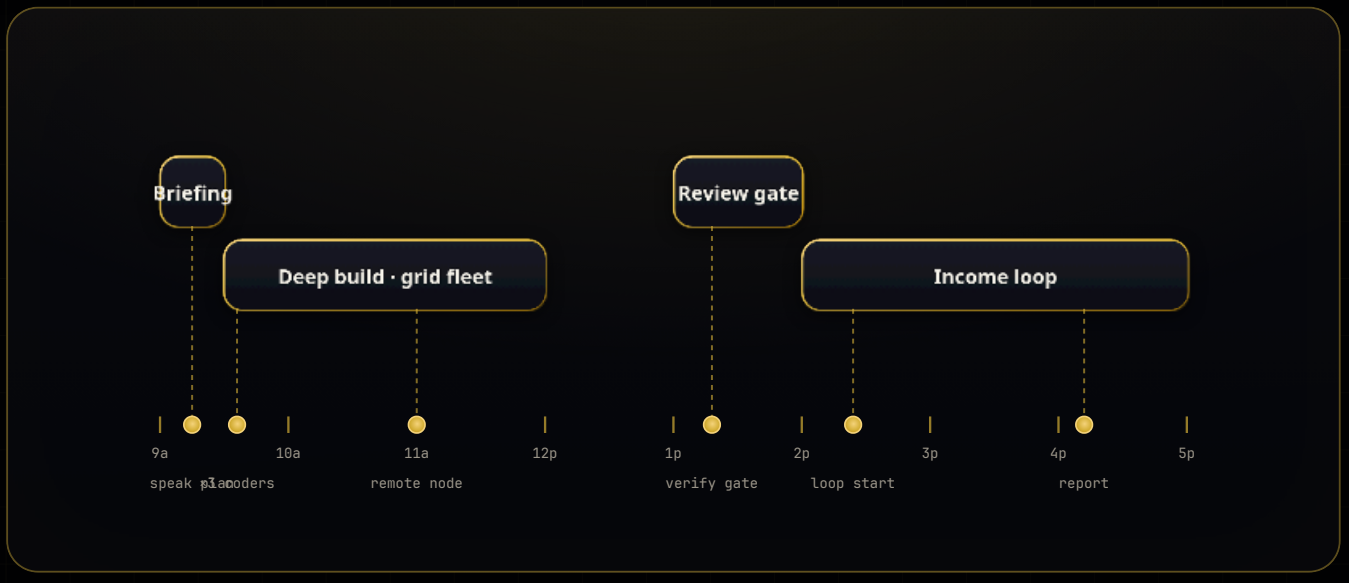
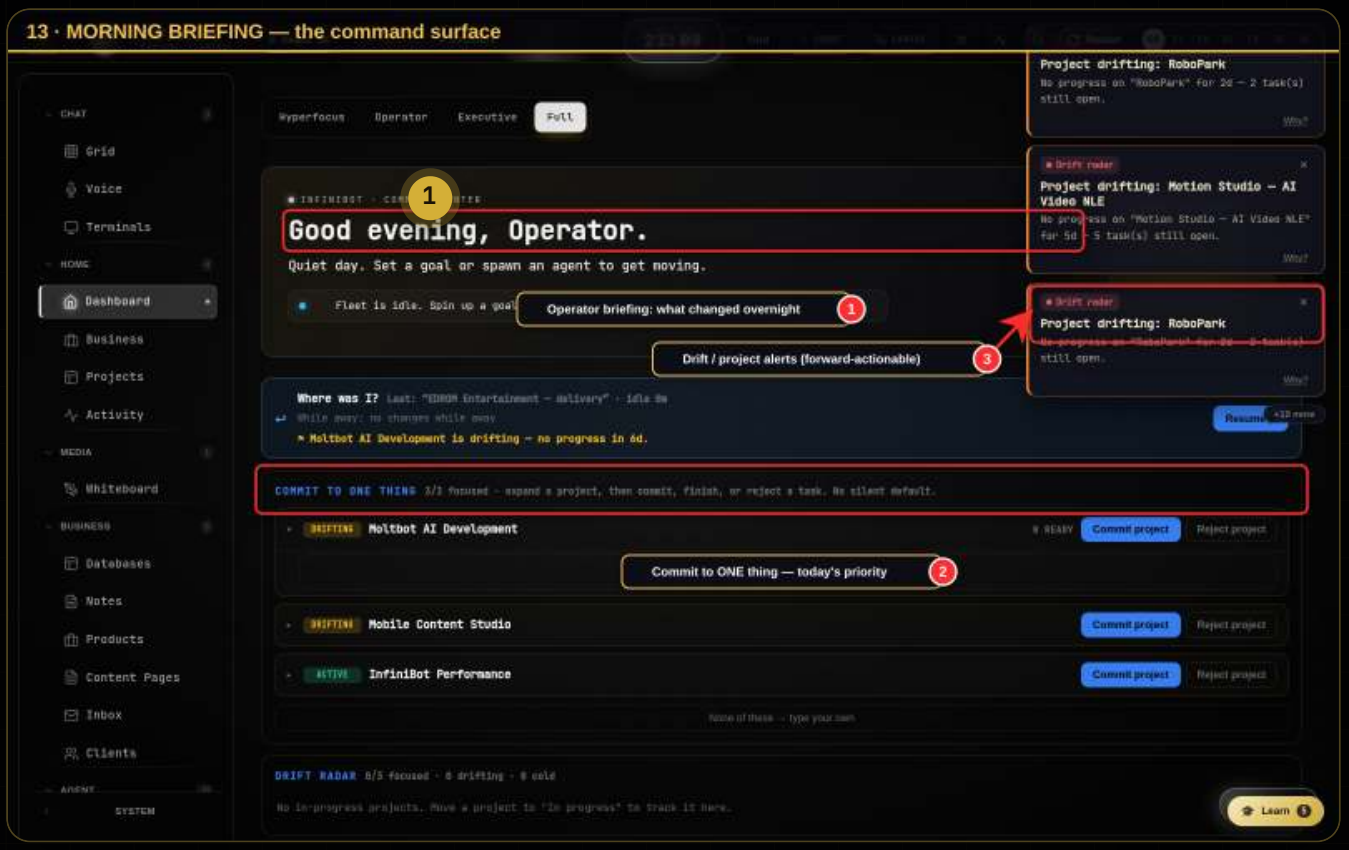


Figure · A day at 10x — focus blocks across an 8-hour axis, with agent-spawn markers showing where the fleet picked up work.

1	∞	10×	0
OPERATOR	PARALLEL AGENTS	THROUGHPUT	DROPPED LOOPS



THE MORNING BRIEFING — WHERE THE 10X DAY STARTS, ALREADY ORIENTED

1. **Morning briefing** (§13) tells you what changed overnight.
2. You **speak** the day's plan (§1); the orchestrator fans it into **parallel coders** (§2) that land as **grid panels** (§1b).
3. Heavy jobs go to a **remote node** (§4); the **kanban board** (§3) is the queue.
4. Recurring chores run as **continuous workflows** (§5) in the background.
5. Each agent carries a **stack of MCP tools** (§6) and pulls context from the **knowledge graph** (§7).
6. **Discipline + proactive alerts** (§8) ping you when something needs a human; you review and gate work in the **focus modal** (§9).
7. The **world map** (§10) and **agent mesh** (§11) keep you oriented; the **credits** page (§12) proves the throughput.

That's the 10x: one operator, a fleet of agents, every loop closed.

> **Next — Part 3:** turning this throughput into income. Productizing your agent workflows, the KinglyVault + Whop license stack, and shipping client work at fleet speed.

All screenshots in `assets/02/` are live captures of the running InfiniBot control terminal, annotated with the exact controls referenced above.

TRY THIS IN INFINIBOT NEXT CHAPTER →

04

PART 04 · MONEY MOVES

Part 3 — End-to-End Income Loops

> A loop you run once earns your first $100. The same loop, run fifty times, is a business.

MONEY MOVES

Part 3 — End-to-End Income Loops

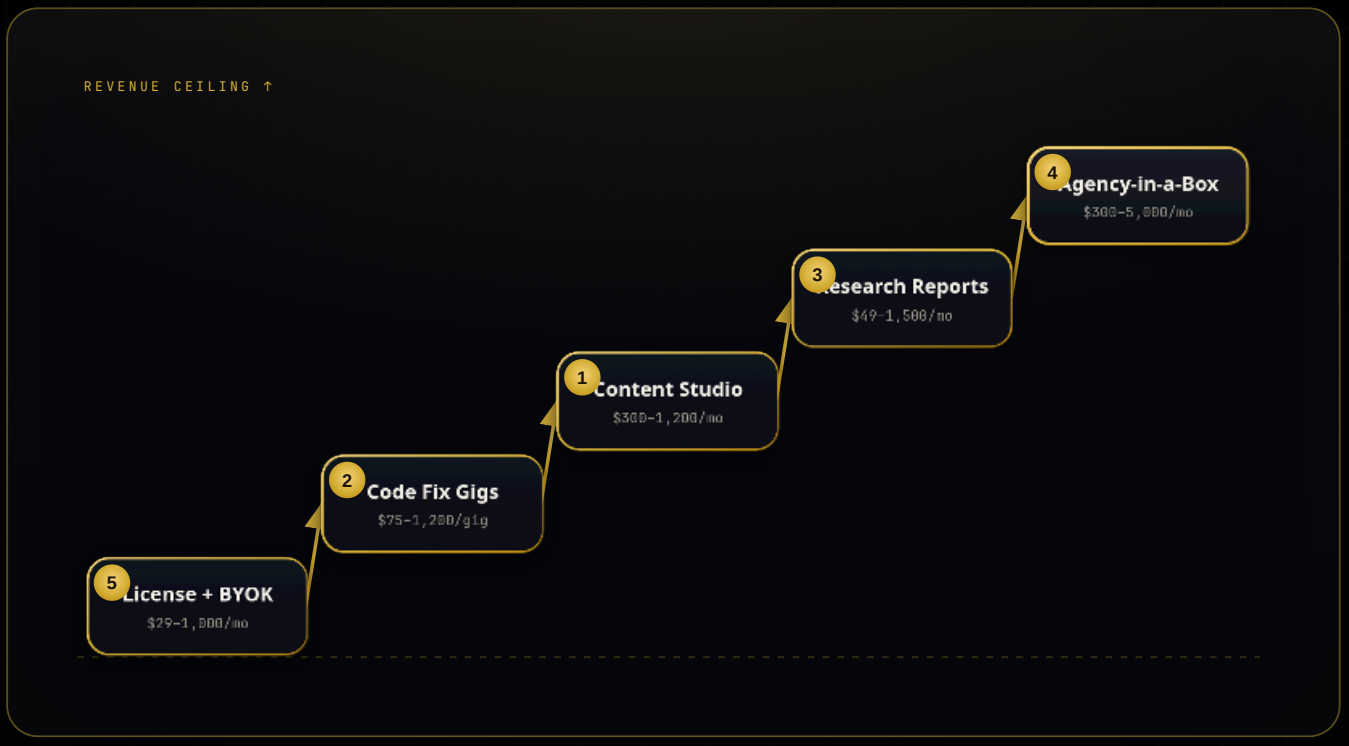
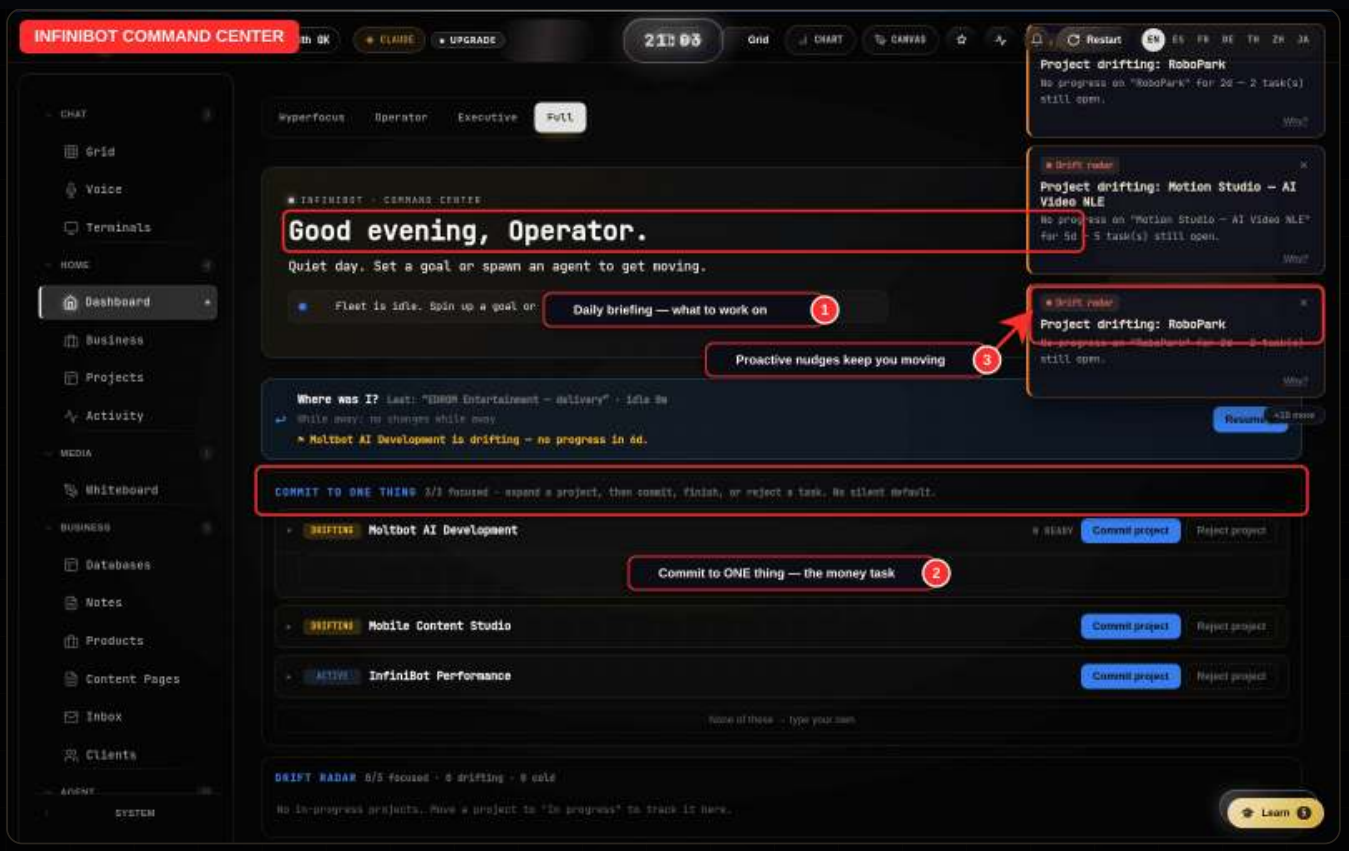


Figure · The income ladder — five repeatable loops, climbing from a first $100 to a managed agency.

> **InfiniBot Beginner-to-Income Playbook · Part 3** Concrete, repeatable money-making pipelines you can run *today* — from idea to delivered revenue. Every step below is shown on the **real InfiniBot UI** (no mockups). Red arrows and numbered circles point at the exact button to press.

An *income loop* is a repeatable pipeline: a **persona** (who you sell to), an **offer** (what they buy), a fixed set of **tools**, a **step-by-step** you can run on autopilot, **pricing** you can defend, a **delivery** mechanism, and a **scale** path. Run the loop once for $100. Run it fifty times for a business.

Everything starts from your command center:



INFINIBOT COMMAND CENTER

1. **Daily briefing** tells you what to work on.
2. **Commit to ONE thing** — pick the single money task for today.
3. **Proactive nudges** keep you moving when you drift.

> **Price everything from real numbers.** Before you quote any client, open `/credits`. It shows your *actual* provider spend so you never sell a job for less than it costs to run. We reference it in every loop below.

Loop 1 — AI Content Studio for Creators

PERSONA
creators · brands

FIRST $100
$300–1,200/mo

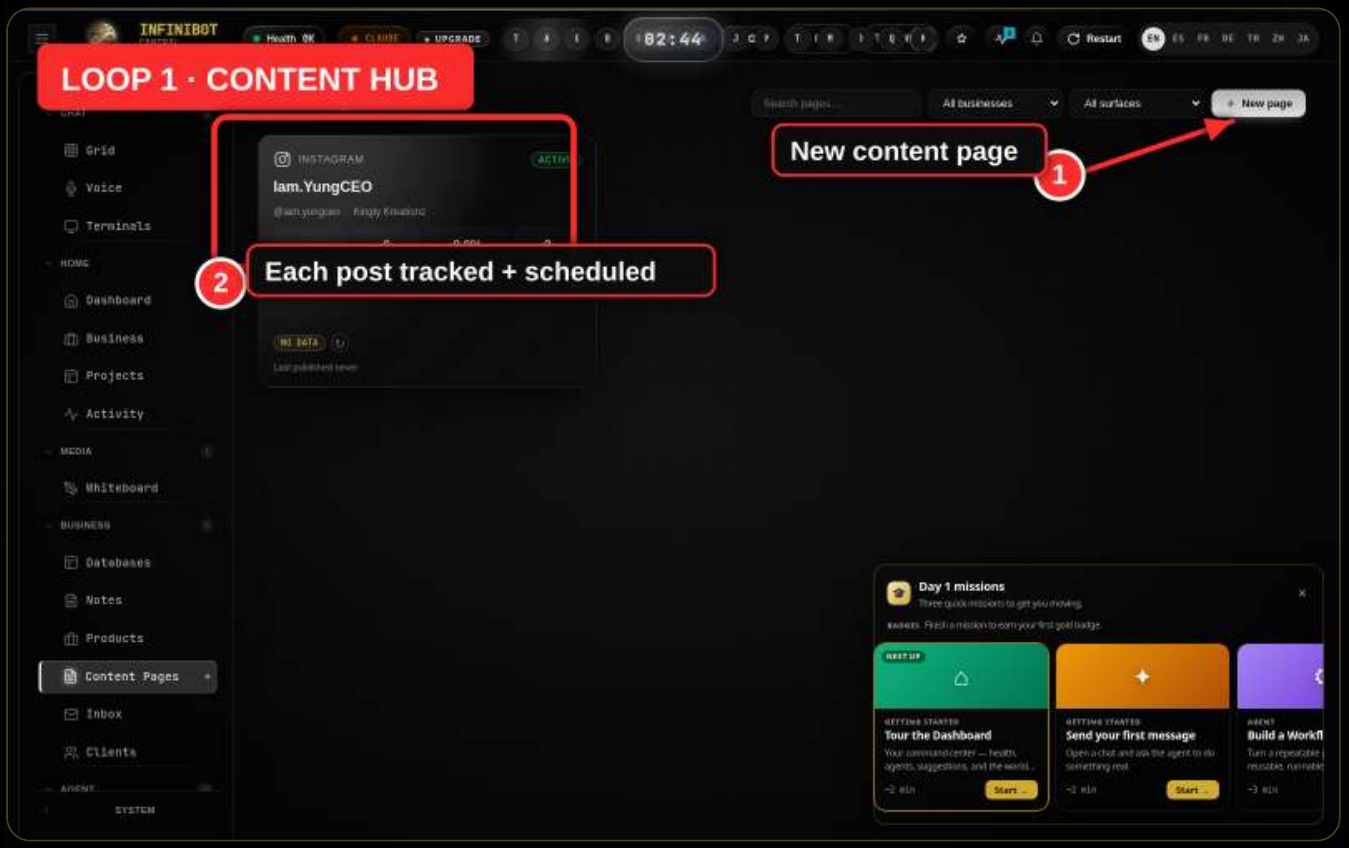
1. **Plan** — Content Hub
2. **Generate** — flypic img/vid
3. **Caption** — content agent
4. **Polish** — Canva
5. **Deliver** — schedule + ship

repeatable pipeline — run once for $100, run fifty times for a business

Figure · Loop 1 — AI Content Studio: plan → generate → caption → polish → deliver.

✅ FIRST $100 IN 7 DAYS

- ✓ Day 1 — Pick ONE niche (e.g. local gyms); create a sample Content Hub page.
- ✓ Day 2 — Generate a 5-post sample pack in flypic (1 niche look).
- ✓ Day 3 — Polish in Canva; post the sample free on your own profile.
- ✓ Day 4–5 — DM 20 creators/brands: sample + “$300/mo, first week free”.
- ✓ Day 6 — Close 1 client; collect a $100 deposit.
- ✓ Day 7 — Deliver the week-1 pack; ask for a testimonial.



CONTENT HUB — ONE TRACKED PAGE PER CLIENT

Persona. Solo creators, coaches, and small e-commerce brands who need a steady stream of posts (Reels/TikTok/Shorts + carousels) but have no time and no design skill.

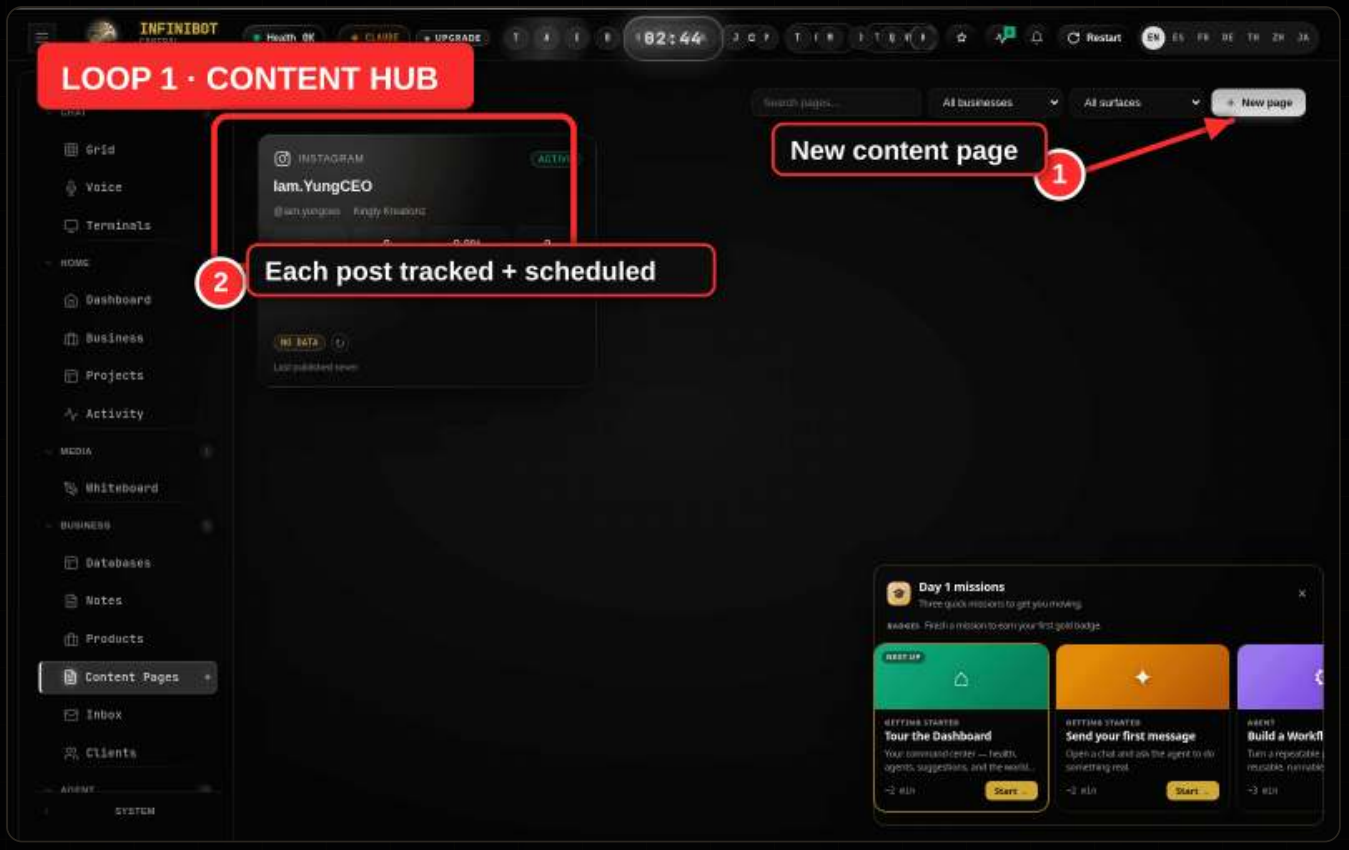
Offer. "Done-for-you content packs": 12 platform-ready posts/week — hook-written captions, AI images, and short video — published on a schedule.

Tools used.

- **Content Hub** (`/content-pages`) — plan and track every post.
- **flypic** (image + video generation) — the visual engine.
- **Canva** — final polish / brand templates (logo, fonts, CTA frames).
- **Chat / content agent** — script + caption generation.

Step-by-step

1. Open the Content Hub and create the client's content page.



CONTENT HUB

- ① Click **New page** to spin up a tracked content page for the client.
- ② Every post lives here as a row — status, platform, and schedule in one place, so nothing slips.

2. Generate the visuals in flypic.

FLYPIC GENERATE PANEL

- ① **Describe the shot** in the prompt bar ("product flat-lay, warm light, minimalist, space for text top-third").
- ② **Pick the aspect ratio** — `9:16` for Reels/Shorts/TikTok, `1:1` for feed, `16:9` for YouTube. Use **Reference** to lock a brand look across a whole pack.
- ③ **Generate**. Repeat for images and short video clips until you have the pack.

3. Write captions + hooks with the content agent. In **Chat**, ask your content agent: *"Write 12 scroll-stopping captions for [brand], each with a hook, 3 value lines, and a CTA."* (This is the same script pipeline Part 2 covered.)

4. Polish in Canva. Drop the flypic visuals into the client's Canva brand template, add logo + CTA frame, and export final assets.

5. Schedule + mark each post back on the Content Hub page so the client can see the whole week at a glance.

Pricing guidance

PACKAGE	WHAT'S INCLUDED	PRICE
Starter	8 posts/week, images only	**$300/mo**
Growth	12 posts/week, images + 2 videos	**$600/mo**
Brand	20 posts/week, video-first, strategy call	**$1,200/mo**

Check `/credits` after your first full pack — generation cost is typically a few dollars, so a $300 starter is ~95%+ margin.

Delivery

Share the Content Hub page (or a Canva folder) with the client; deliver weekly on a fixed day. Keep one page per client so renewals are one click.

How to scale

- Build **3–4 reusable flypic Reference looks** per niche → packs get faster.
- Turn the caption prompt into a saved workflow.
- Hire a VA to publish from the Content Hub while you only run generation.

✅ First $100 in 7 days

- ☐ **Day 1:** Pick ONE niche (e.g. local gyms). Create a sample Content Hub page.
- ☐ **Day 2:** Generate a 5-post sample pack in flypic (1 niche look).
- ☐ **Day 3:** Polish in Canva, post the sample free on your own profile.
- ☐ **Day 4–5:** DM 20 creators/brands the sample + "$300/mo, first week free".
- ☐ **Day 6:** Close 1 client; collect a **$100 deposit**.
- ☐ **Day 7:** Deliver week-1 pack. Ask for a testimonial.

Loop 2 — Done-for-You Code Fix Gigs

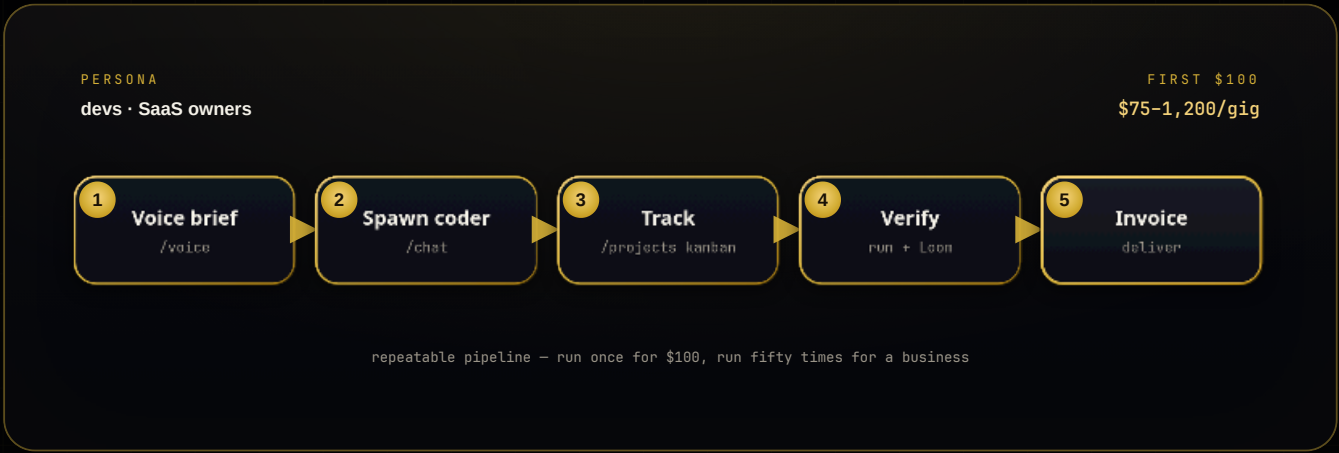
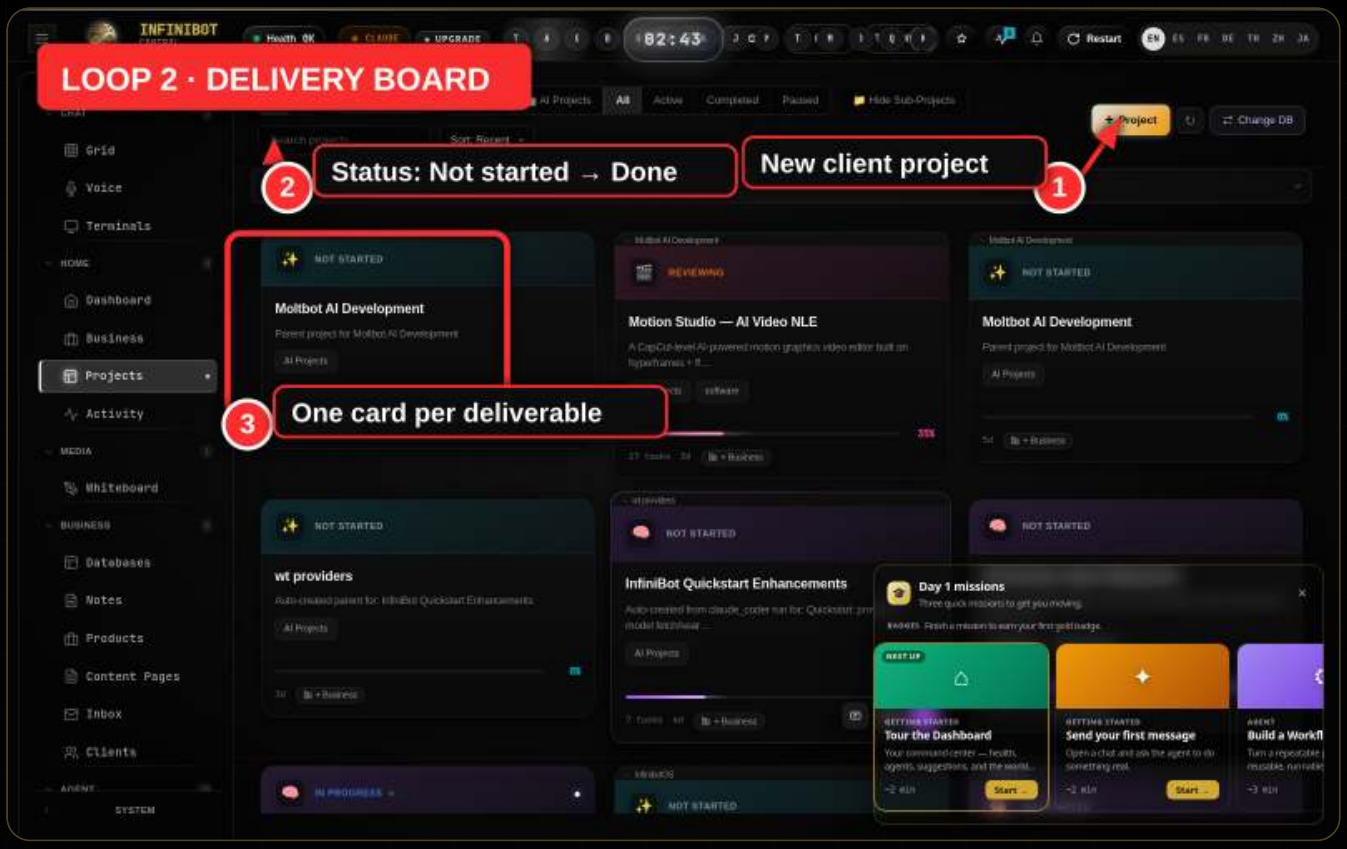


Figure · Loop 2 — Done-for-You Code Fixes: voice brief → spawn → track → verify → invoice.

✅ FIRST $100 IN 7 DAYS

- ☑ Day 1 — Post in 2 indie-hacker / freelance communities: “24h bug fixes, $150”.
- ☑ Day 2 — Take 1 free/cheap fix for a testimonial; run it via voice + chat.
- ☑ Day 3 — Deliver with a Loom + Kanban card; ask for a referral.
- ☑ Day 4–6 — Pitch 15 SaaS owners with the testimonial; book 1 paid gig.

- [x] Day 7 — Deliver the paid gig → $150 collected.



THE DELIVERY BOARD — ONE CARD PER DELIVERABLE

Persona. Indie hackers, agencies, and small SaaS owners with a broken build, a stubborn bug, or a "small feature" they can't ship.

Offer. Fixed-scope code fixes delivered in 24–48h with a screen-recorded walkthrough and an invoice. You drive it *by voice* and let a coder agent do the work.

Tools used.

- **Voice** (`/voice`) — spawn and steer agents hands-free.
- **Chat** (`/chat`) — paste the brief, spawn the coder agent, watch it work.
- **Projects / Kanban** (`/projects`) — the delivery board the client can see.
- A screen recorder (OBS / Loom) + any invoicing tool.

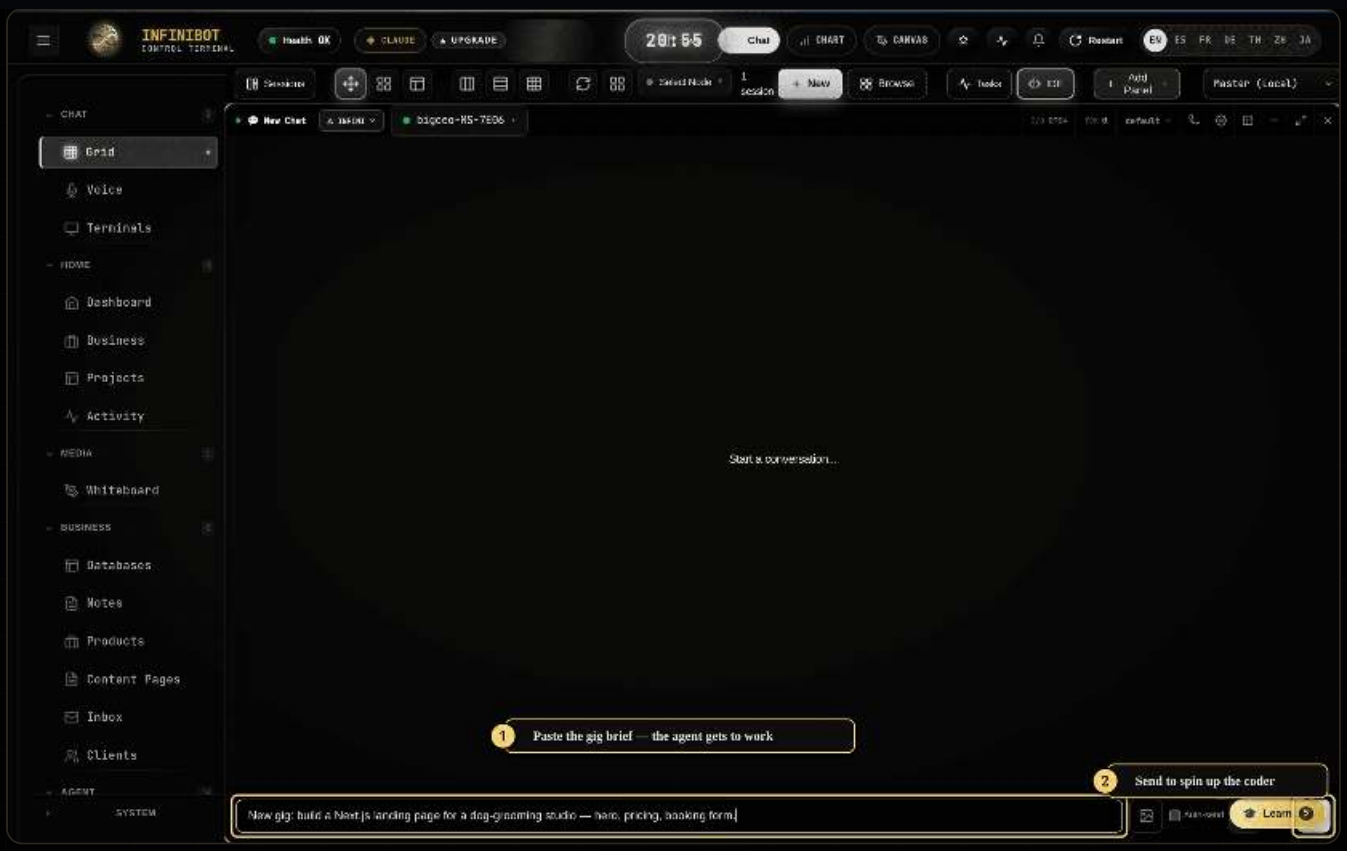
Step-by-step

1. Kick it off by voice.

VOICE ORCHESTRATOR

- ① The **live voice orchestrator** is your hands-free control.
- ② Say it out loud: *"Fix the failing build in the checkout repo and add a unit test."* The orchestrator spins up a coder agent for you.

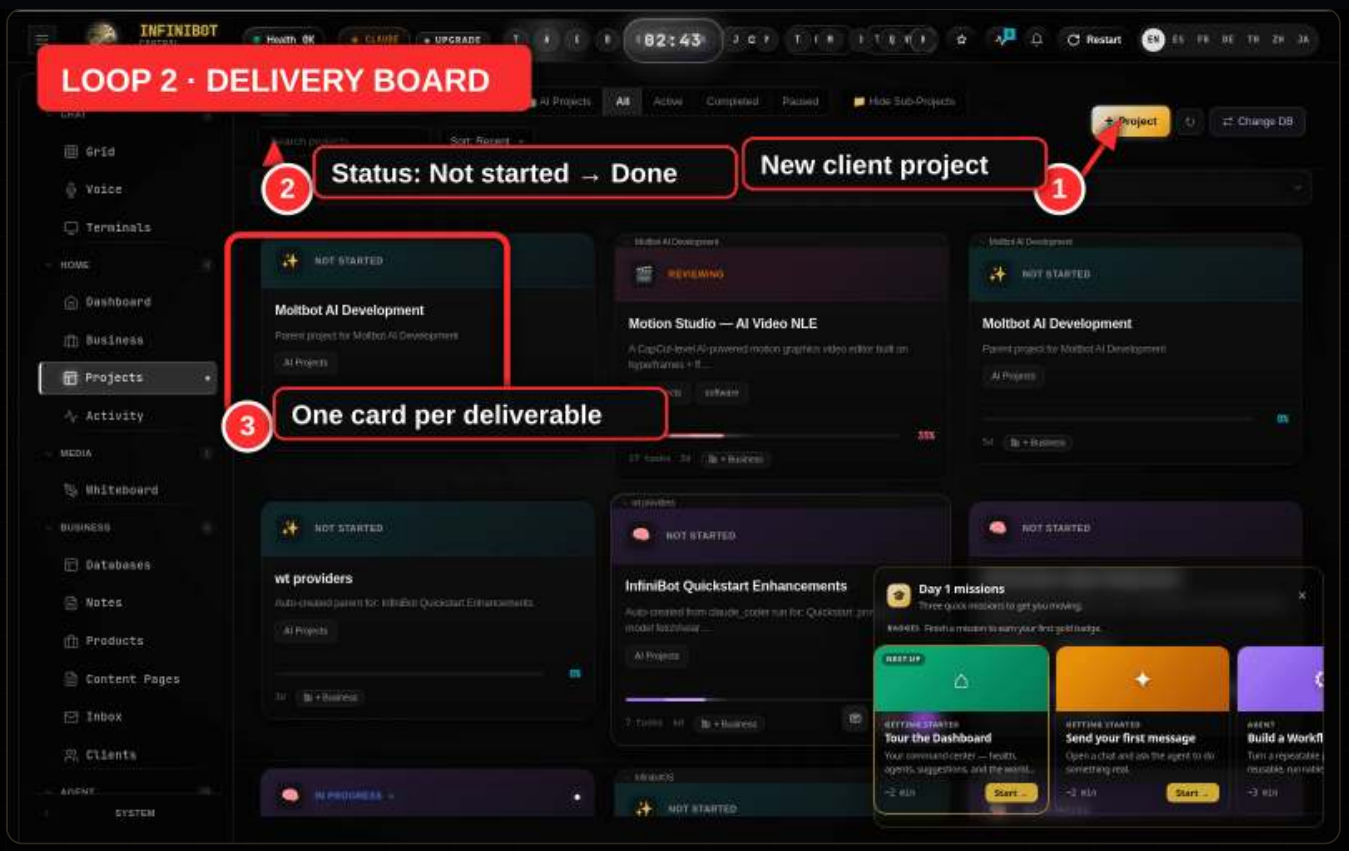
2. Drop the gig brief in Chat and spawn the coder agent.



SPAWN A CODER AGENT

- ① **Paste the gig brief** (repo, the bug, the acceptance criteria) into the message box and send.
- ② Agents run **live** — you can watch progress in real time. Get the brief *right the first time*; mid-run "corrections" often don't reach the worker.

3. Track delivery on the Kanban board.



DELIVERY BOARD

- ① **New client project** — one project per gig.
- ② Move it through **Not started** → **In progress** → **Done** so the client always knows where things stand.
- ③ **One card per deliverable** (the fix, the test, the docs). This board *is* your status report.

4. Verify, then screen-record. Confirm the fix actually works (run it), then record a 2–3 minute Loom: the bug, the fix, the passing test.

5. Send the recording + invoice. Attach the walkthrough, mark the card Done.

Pricing guidance

TIER	SCOPE	PRICE
Quick fix	1 bug, < 2h	**$75–150**
Standard	bug + test + short walkthrough	**$250–400**
Feature	small scoped feature, 1–2 days	**$600–1,200**

Open `/credits` to see the agent run cost (usually a few dollars). Price on *value and turnaround*, not tokens — a same-day fix is worth far more than its compute.

Delivery

The Kanban card + Loom video + invoice. Keep the repo branch/PR link on the card.

How to scale

- Run multiple gigs in parallel — each is its own project + agent.
- Productize the top 3 repeat requests ("CI green in 24h", "add Stripe checkout").
- Offer a **$500/mo retainer**: "X fixes per month, 48h SLA".

✅ First $100 in 7 days

- ☐ **Day 1:** Post in 2 indie-hacker / freelance communities: "24h bug fixes, $150".
- ☐ **Day 2:** Take 1 free/cheap fix to get a testimonial; run it via voice + chat.
- ☐ **Day 3:** Deliver with a Loom + Kanban card. Ask for a referral.
- ☐ **Day 4–6:** Pitch 15 SaaS owners with the testimonial; book 1 paid gig.
- ☐ **Day 7:** Deliver the paid gig → **$150 collected**.

Loop 3 — Automated Research Reports (Subscription)

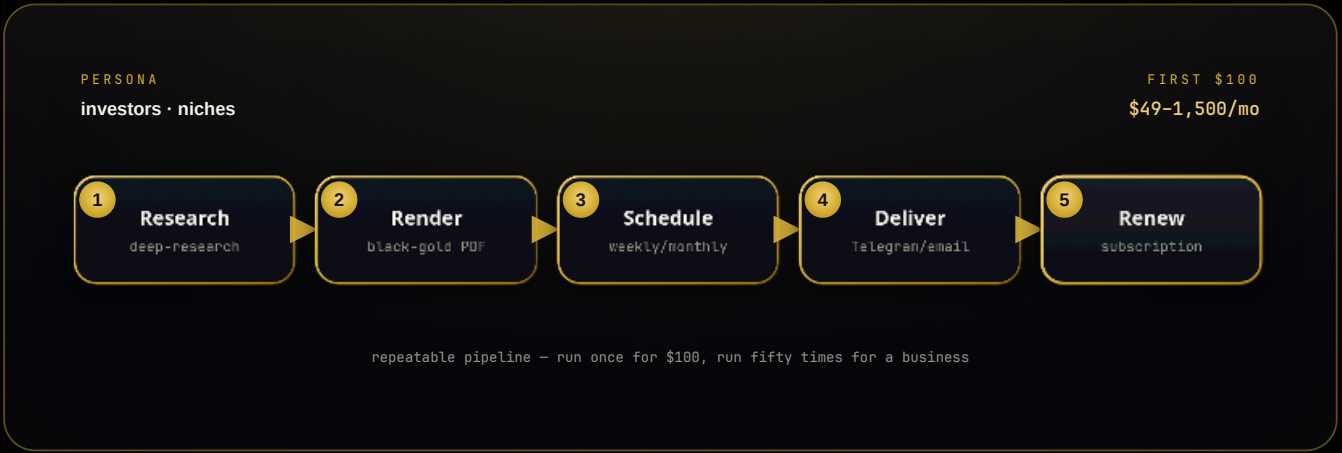
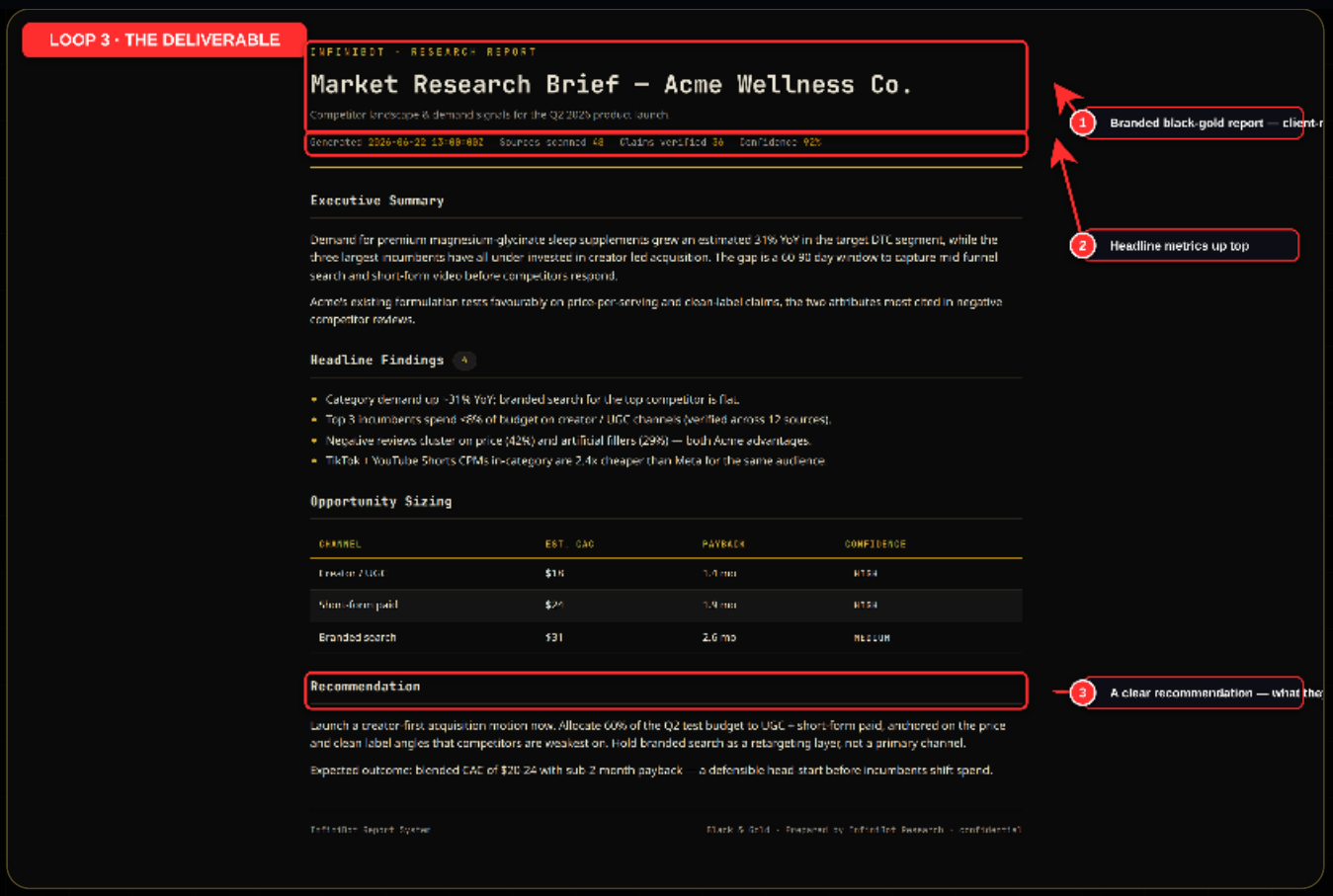


Figure · Loop 3 — Automated Research Reports: research → render → schedule → deliver → renew (MRR).

✅ FIRST $100 IN 7 DAYS

- ☑ Day 1 — Pick a topic with a paying audience (a niche, a ticker, an industry).
- ☑ Day 2 — Run one full research sweep; render the black-gold PDF.
- ☑ Day 3 — Post the report free in the relevant community as a sample.
- ☑ Day 4–5 — Offer "weekly version, $49/mo" to everyone who engaged.
- ☑ Day 6 — Close 2–3 subscribers → $100+ MRR.
- ☑ Day 7 — Set the Schedule so issue #2 ships itself.



THE DELIVERABLE — A BRANDED BLACK-GOLD REPORT PDF

Persona. Investors, founders, consultants, and niche communities who need to stay current but won't read 40 sources a week.

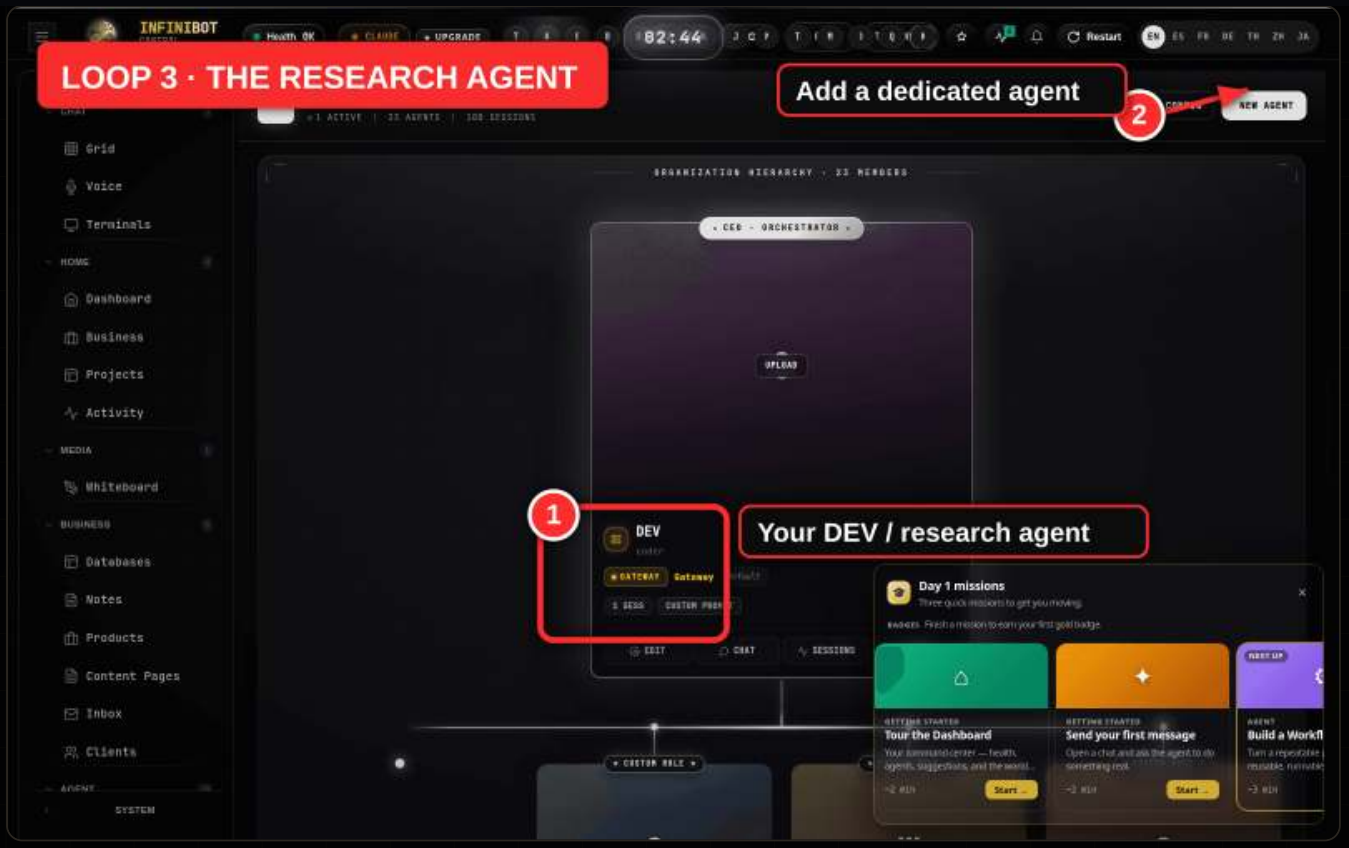
Offer. A recurring branded intelligence report — weekly or monthly — on a topic they care about. Looks like a $5k consultancy deliverable; runs on autopilot.

Tools used.

- **Research** (deep-research / research MCP) — multi-source, fact-checked.
- **Agents** (`/agents`) — your dedicated research agent.
- **Reports** (`/reports`) — render the canonical **black-gold PDF** + schedule it.
- **Telegram / email** — automated delivery.

Step-by-step

1. Stand up your research agent.

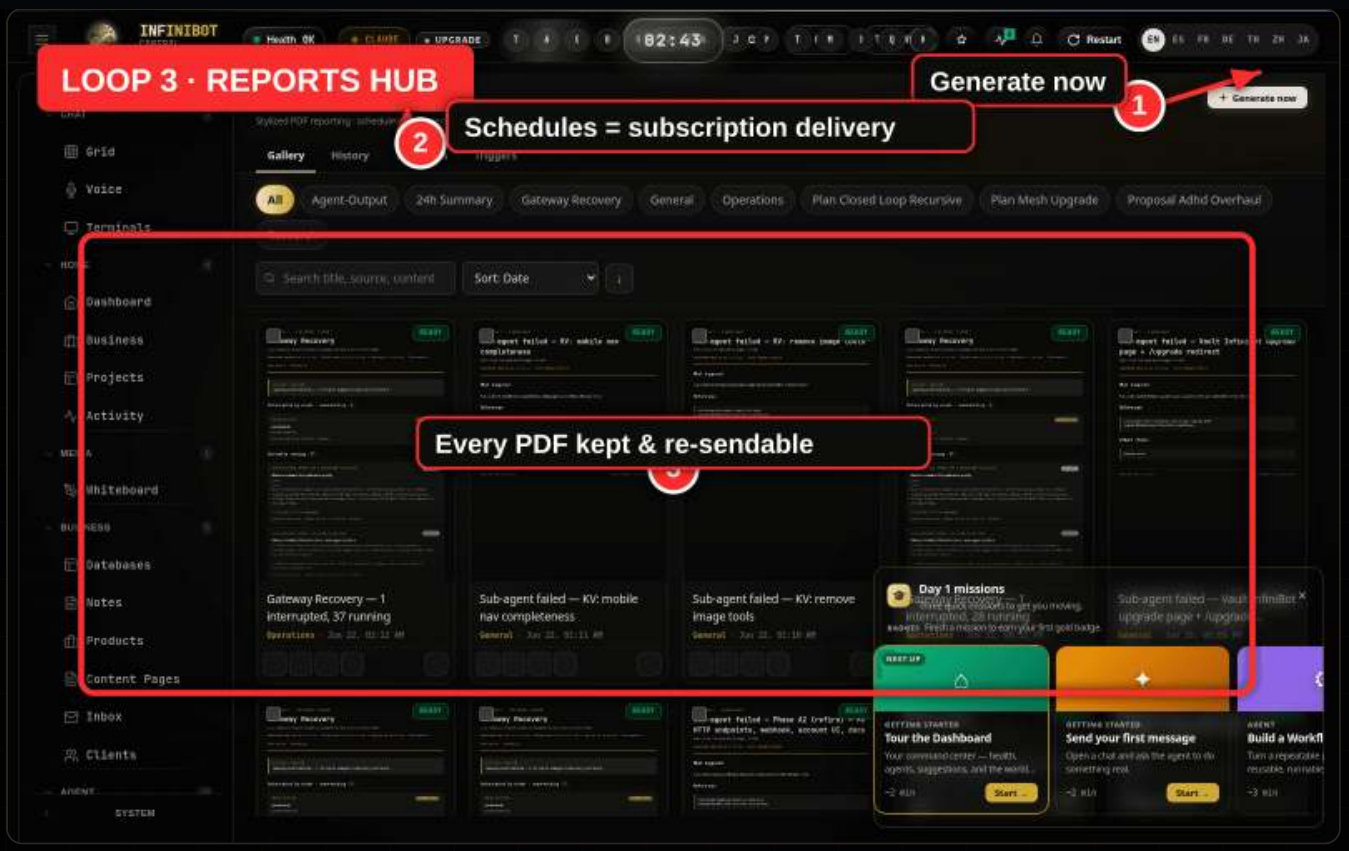


RESEARCH AGENT

- ① Your **DEV / research agent** is the worker that runs the deep-research sweep.
- ② Need a focused one per client topic? **Add a dedicated agent**.

2. Run the research sweep. Hand the agent the brief: *"Weekly digest on AI voice agents — fan out searches, verify claims, synthesize with sources."* The deep-research harness fetches sources, adversarially verifies, and synthesizes.

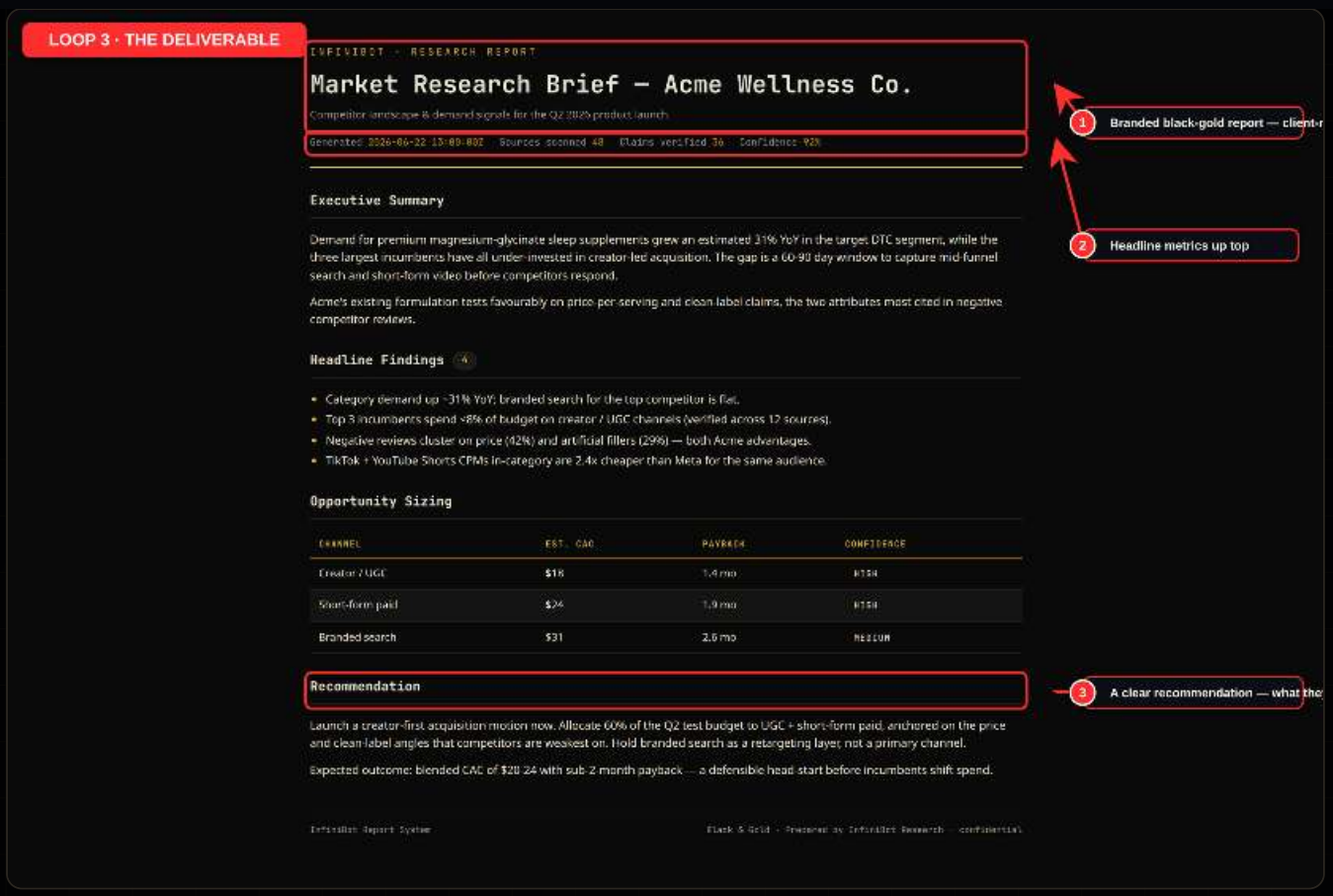
3. Render the report and schedule delivery.



REPORTS HUB

- ① **Generate now** to render a one-off, or…
- ② open **Schedules** — this is what turns a report into a **subscription** (auto-render weekly/monthly).
- ③ Every PDF is **kept and re-sendable** from the gallery.

4. The deliverable. This is the actual canonical InfiniBot report PDF:



THE DELIVERABLE REPORT PDF

- ① **Branded black-gold report** — premium, consistent, client-ready.
- ② **Headline metrics** up top (sources scanned, claims verified, confidence).
- ③ A **clear recommendation** — that's what they actually pay for.

5. Deliver automatically to Telegram or email on the schedule. The client gets it without you lifting a finger.

Pricing guidance

PLAN	CADENCE	PRICE
Monthly brief	1 report/mo	**$49–99/mo**
Weekly brief	4 reports/mo	**$199–299/mo**
Private / bespoke topic	weekly, custom scope	**$500–1,500/mo**

Subscriptions = predictable MRR. Check `/credits` : a verified research run is a few dollars, so even a $49/mo plan is strongly profitable — and it compounds.

Delivery

Scheduled PDF to Telegram/email. Keep the `/reports` gallery as the client's archive of past issues.

How to scale

- One topic → many subscribers (same report, sell N times).

- Add tiers: raw digest vs. digest + a 15-min Loom commentary.
- Spin up one scheduled agent per topic and let them run unattended.

✅ First $100 in 7 days

- ☐ **Day 1:** Pick a topic with a paying audience (a niche, a ticker, an industry).
- ☐ **Day 2:** Run one full research sweep; render the black-gold PDF.
- ☐ **Day 3:** Post the report free in the relevant community as a sample.
- ☐ **Day 4–5:** Offer "weekly version, $49/mo" to everyone who engaged.
- ☐ **Day 6:** Close 2–3 subscribers → **$100+ MRR**.
- ☐ **Day 7:** Set the **Schedule** so issue #2 ships itself.

Loop 4 — Agency-in-a-Box (Federation Hosting)

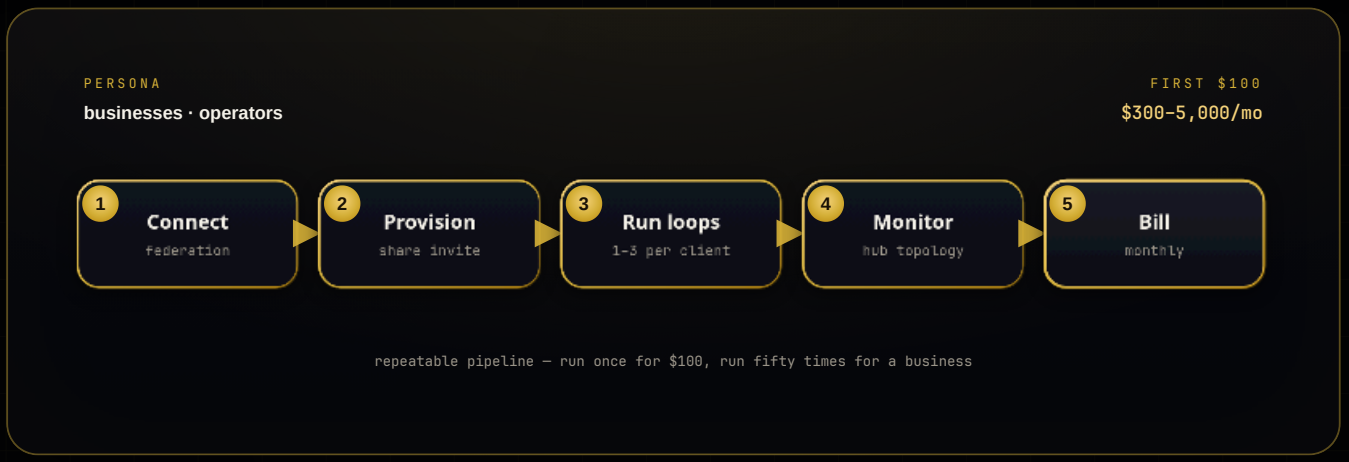
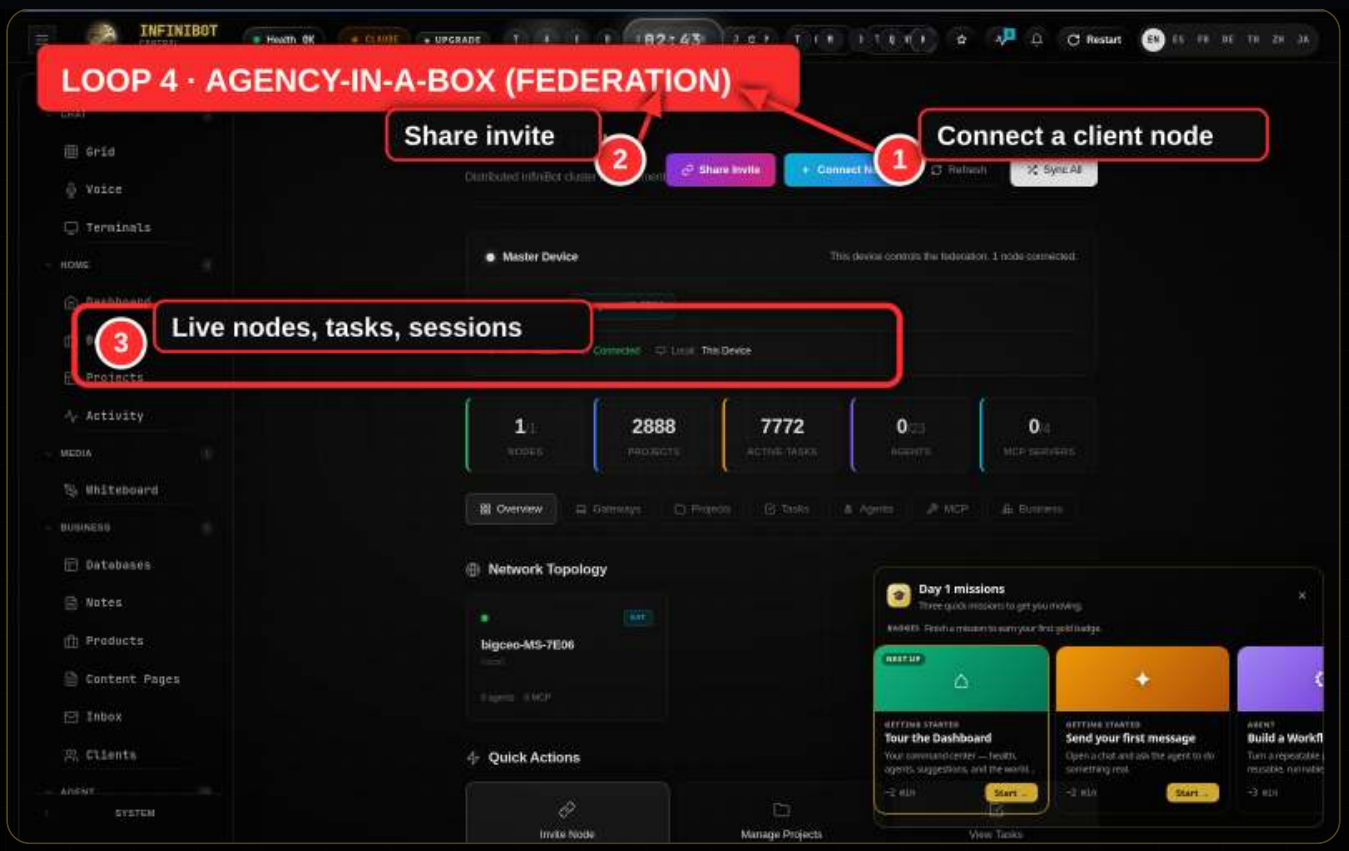


Figure · Loop 4 — Agency-in-a-Box: connect node → provision → run loops → monitor → bill.

✅ FIRST $100 IN 7 DAYS

- ☑ Day 1 — Set up your master hub; do a test connect of a second node.
- ☑ Day 2 — Write a one-page “Your own AI agency, managed” offer.
- ☑ Day 3–5 — Pitch 10 local businesses / operators you already know.
- ☑ Day 6 — Onboard 1 via the share-invite link; collect a $100 setup fee.
- ☑ Day 7 — Configure their first loop; convert to monthly.



FEDERATION NETWORK HUB — EVERY CLIENT NODE, ONE SCREEN

Persona. Small businesses and other operators who want their *own* AI agent fleet but can't set up or maintain one. You host it for them.

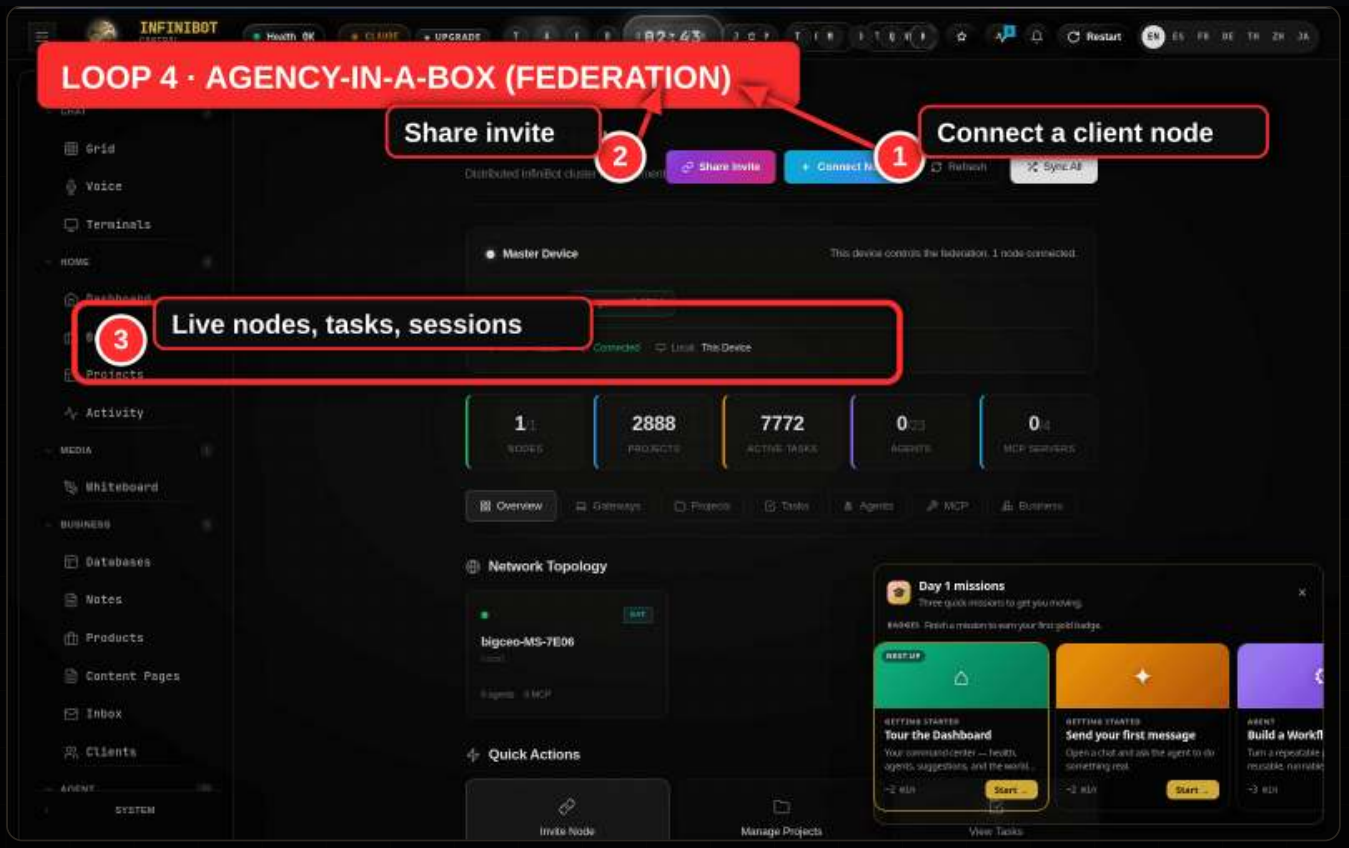
Offer. A managed InfiniBot node connected to your hub: their agents, their data, your maintenance — billed monthly. You're the MSP for AI agents.

Tools used.

- **Federation** (`/federation`) — connect, monitor, and manage client nodes.
- The whole InfiniBot stack (agents, voice, content, reports) delivered *per client*.

Step-by-step

1. Open the Federation Network Hub.



FEDERATION NETWORK HUB

- ① **Connect a client node** to bring their machine/VPS into your network.
- ② **Share invite** to onboard them with a one-time link (one-command setup).
- ③ Watch **live nodes, tasks, and sessions** from your master device — this is your operations dashboard across every client.

2. Provision the client. From the invite, the client runs the satellite-setup one-liner; their node appears under your hub (master ↔ satellite, hub-and-spoke routing).

3. Stand up their loops. Configure the income loops *they* want on their node — content, code fixes, research reports — using Loops 1–3 above.

4. Monitor + maintain. Use the hub's topology, tasks, and sessions tabs to keep every client healthy. Updates roll out from the master.

5. Bill monthly. Hosting + management + per-loop add-ons.

Pricing guidance

TIER	WHAT YOU MANAGE	PRICE
Managed node	hosting + updates + monitoring	**$300–500/mo**
Node + 1 loop	above + run one income loop for them	**$800–1,500/mo**
Full agency	node + multiple loops + SLA	**$2,000–5,000/mo**

Your cost is the node's compute + your time. Track each client's spend in `/credits` and pass it through (or mark it up) transparently.

Delivery

A live, monitored node + a monthly report (use Loop 3!) summarizing what their fleet did. The Federation hub *is* your proof-of-work.

How to scale

- Standardize onboarding with the share-invite link → minutes per client.
- Template the per-client loop configs.
- Tier by number of nodes; hire an ops person once you pass ~10 nodes.

✅ First $100 in 7 days

- ☐ **Day 1:** Set up your master hub; do a test connect of a second node.
- ☐ **Day 2:** Write a one-page "Your own AI agency, managed" offer.
- ☐ **Day 3–5:** Pitch 10 local businesses / operators you already know.
- ☐ **Day 6:** Onboard 1 via the share-invite link; collect a **$100 setup fee**.
- ☐ **Day 7:** Configure their first loop; convert to monthly.

Loop 5 — License Resale + BYOK Consulting

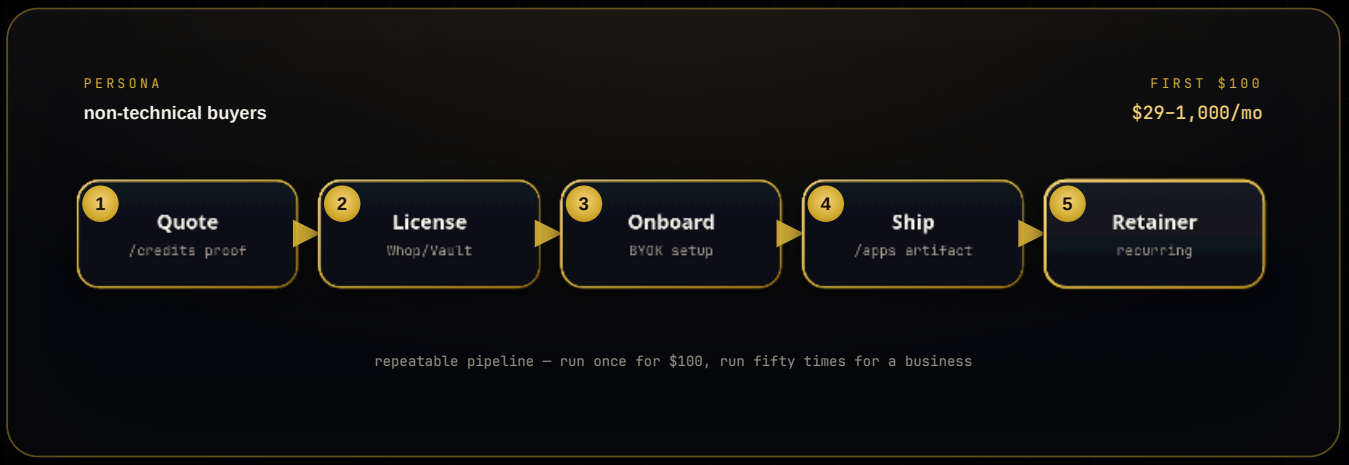
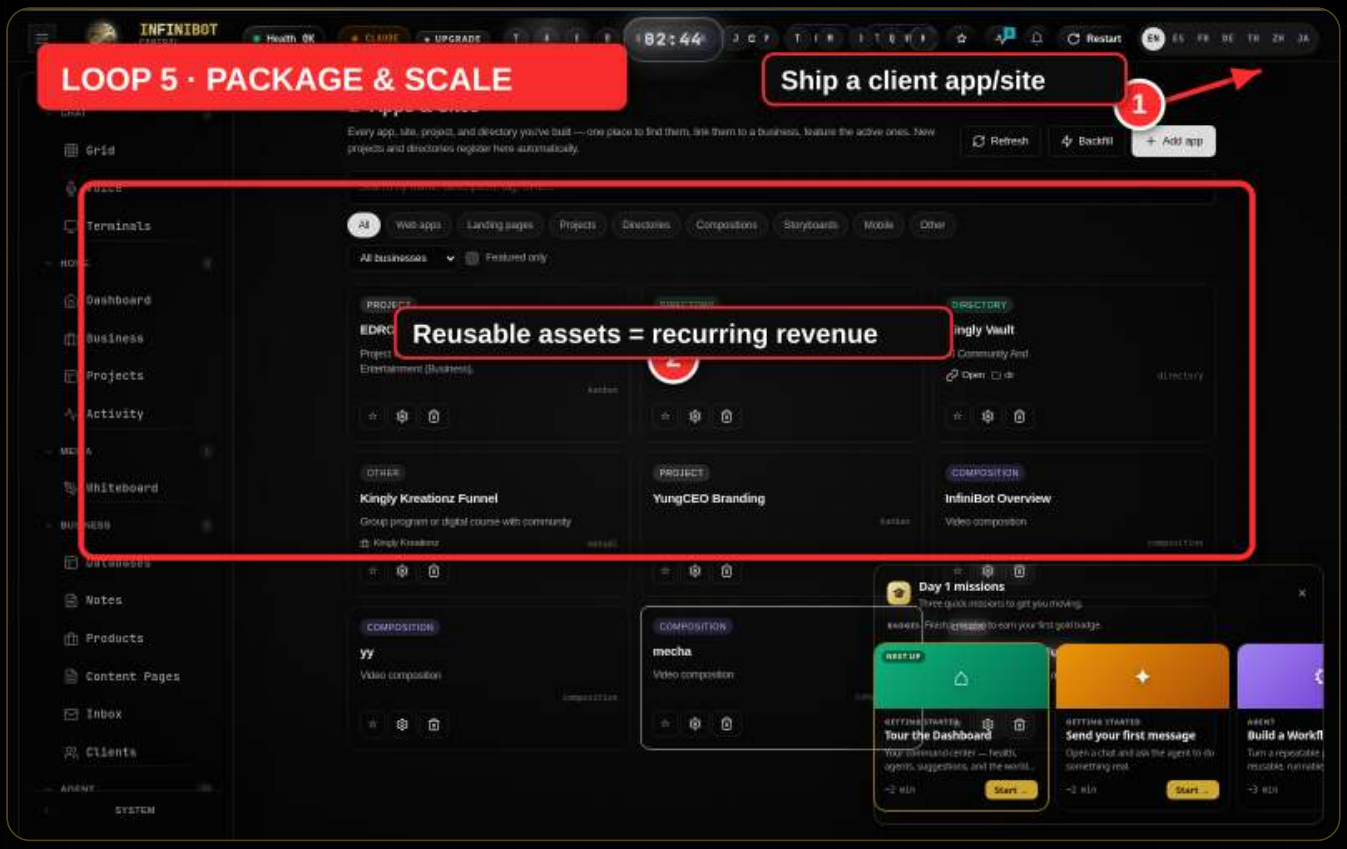


Figure · Loop 5 — License Resale + BYOK Consulting: quote → license → onboard → ship → retainer.

✅ FIRST $100 IN 7 DAYS

- ☑ Day 1 — Set up your Whop/KinglyVault checkout and a simple offer page.
- ☑ Day 2 — Record a 5-min “what InfiniBot does for you” demo.
- ☑ Day 3–5 — Pitch 15 non-technical operators; lead with “you keep your own keys”.
- ☑ Day 6 — Sell 1 BYOK setup at $250 (or 4 licenses at $29) → $100+.

- [x] Day 7 — Run the onboarding; pitch the monthly retainer.



APPS & SITES — SHIP A TANGIBLE ARTIFACT OF THE SETUP

Persona. People who want InfiniBot's power but won't (or can't) set it up, choose models, or wire up their own API keys.

Offer. Two stacked revenue streams:

1. **Resell access** via KinglyVault-issued **Whop** licenses (recurring).
2. **BYOK setup + consulting** — a paid onboarding where you configure their providers, models, voice, and first loops on *their* keys.

> InfiniBot monetizes through **KinglyVault + Whop licenses** plus a login gate, and users **bring their own LLM keys (BYOK)** — so there's no per-user inference cost to you. That's what makes resale + consulting pure margin.

Tools used.

- `/credits` — show clients exactly what their stack costs (and prove BYOK value).
- **Apps & Sites** (`/apps`) — package and hand over what you build.
- Whop / KinglyVault — license issuance + checkout.

Step-by-step

1. Quote and prove value with `/credits`.

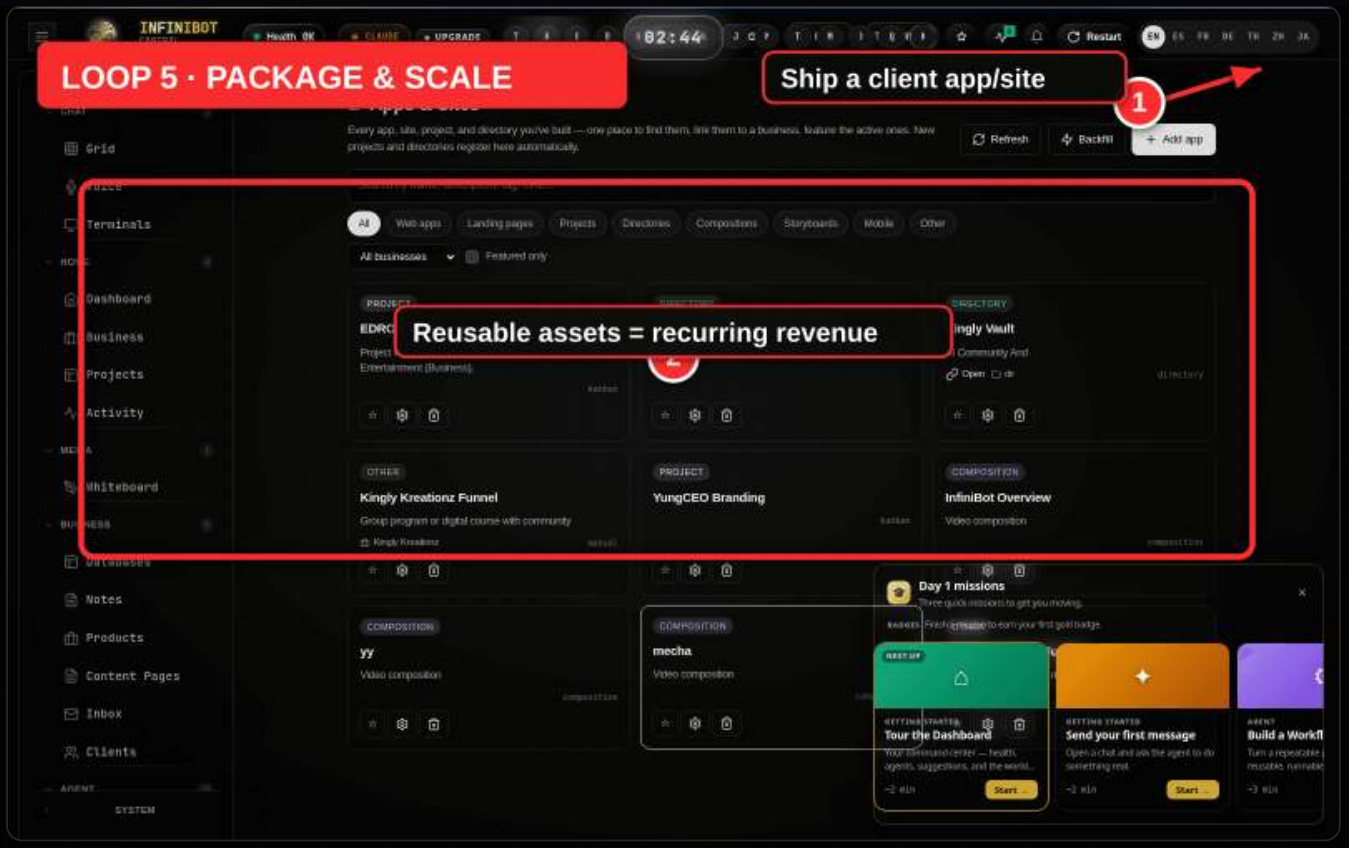
PRICE WITH /CREDITS

- ① **Real provider spend** per backend — this is your honest cost basis.
- ② **Tokens & $ this month** — the headline numbers clients want to see.
- ③ **Cost by backend** — use it to set your consulting margin and show BYOK savings (they pay providers directly, you charge for expertise).

2. Issue the license. Sell access through your Whop/KinglyVault checkout; the login gate handles entitlement.

3. Run the BYOK onboarding. On a call (or via **Voice**), walk them through adding their own API keys, picking models, and enabling voice. They own the keys; you own the know-how.

4. Ship their first deliverables.



PACKAGE & SCALE VIA APPS

- ① **Ship a client app/site** (`/apps`) as a tangible artifact of the setup.
- ② **Reusable assets = recurring revenue** — the apps, configs, and loops you build become the basis for ongoing support.

5. Convert to a retainer. Monthly support + new loops keeps the relationship (and the license) alive.

Pricing guidance

STREAM	OFFER	PRICE
License resale	access via your Whop	**$29–99/mo** (recurring)
BYOK setup	one-time onboarding call + config	**$250–750 one-time**
Consulting retainer	monthly support + new loops	**$300–1,000/mo**

Because of BYOK there's **no inference cost to you** — resale + consulting is near-pure software margin.

Delivery

License access + a recorded onboarding + the apps/configs from `/apps` . Provide a short "how your stack works" doc so they feel ownership.

How to scale

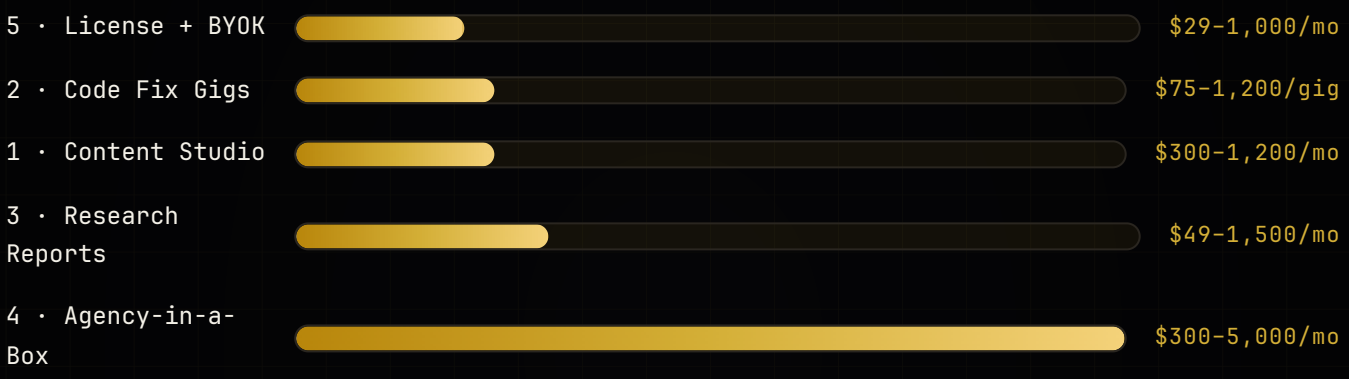
- Build a repeatable onboarding checklist → onboard in under an hour.
- Bundle resale + setup + retainer into one "get started" package.
- Recruit affiliates to resell your Whop licenses for a cut.

✅ First $100 in 7 days

- ☐ **Day 1:** Set up your Whop/KinglyVault checkout and a simple offer page.
- ☐ **Day 2:** Record a 5-min "what InfiniBot does for you" demo.
- ☐ **Day 3–5:** Pitch 15 non-technical operators; lead with "you keep your own keys".
- ☐ **Day 6:** Sell 1 BYOK setup at **$250** (or 4 licenses at $29) → **$100+**.
- ☐ **Day 7:** Run the onboarding; pitch the monthly retainer.

Putting it together

THE INCOME LADDER



◆ Start with whichever loop matches a skill you already have, bank the first $100, then run the other loops **on your own business**. InfiniBot is both the factory and the product.

LOOP	BEST FOR	REVENUE SHAPE	FASTEST TO $100
1 · Content Studio	creators, brands	monthly retainer	deposit on a pack
2 · Code Fix Gigs	devs, SaaS	per-gig → retainer	one paid fix
3 · Research Reports	investors, niches	subscription / MRR	a few subscribers
4 · Agency-in-a-Box	businesses	high monthly	setup fee
5 · License + BYOK	non-technical buyers	recurring + one-time	one setup or 4 licenses

The meta-loop: start with whichever matches a skill you already have, get the first $100, then *use the other loops on your own business* — research your market (Loop 3), make your content (Loop 1), and host it all on your hub (Loop 4). InfiniBot is both the factory and the product.

> **Always price from `/credits`.** Costs change as you scale and switch models. Re-check before every new quote so your margin never quietly erodes.

All screenshots in this guide were captured from a live InfiniBot instance and annotated programmatically (red arrows + numbered steps). Your screens will show your own data, but the buttons are in the same places.

TRY THIS IN INFINIBOT

NEXT CHAPTER →

05

PART 05 · SCALING

Scaling: From One Operator to a Fleet

> One operator, many machines — federation turns a laptop into a fleet you own outright.

SCALING

Scaling: From One Operator to a Fleet

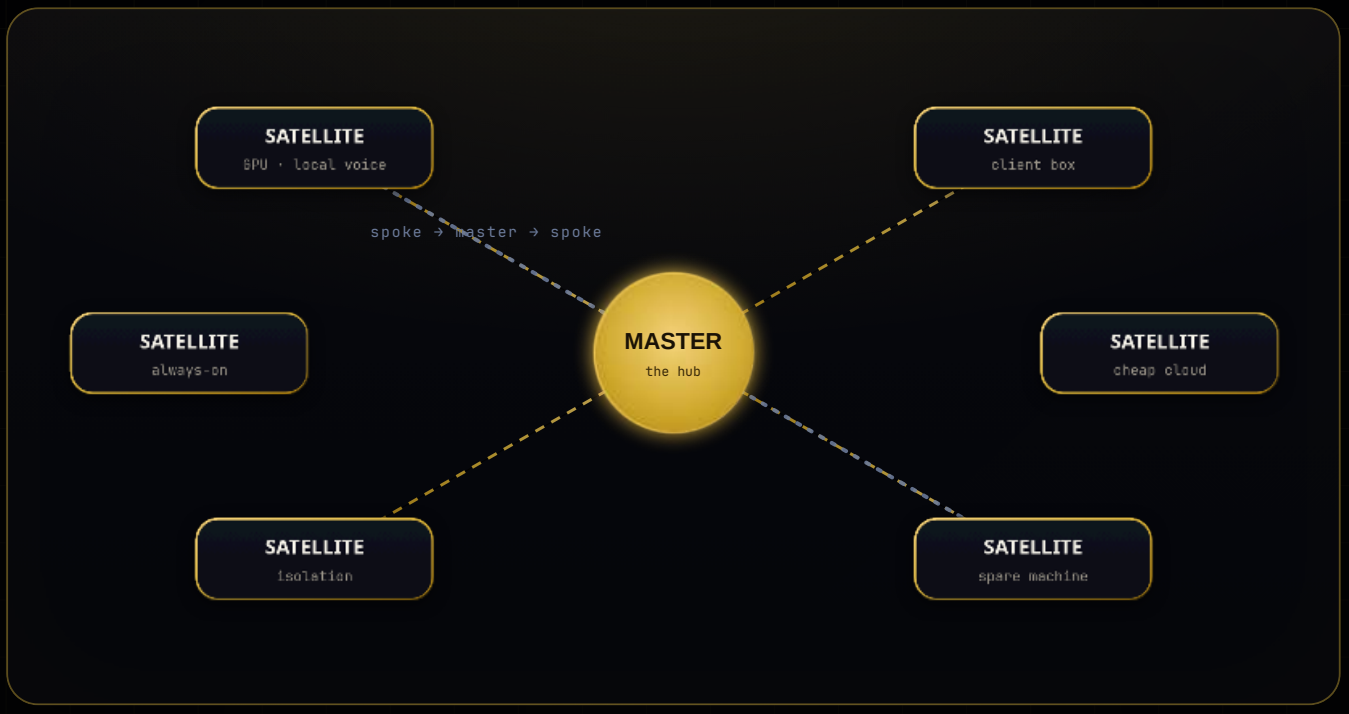
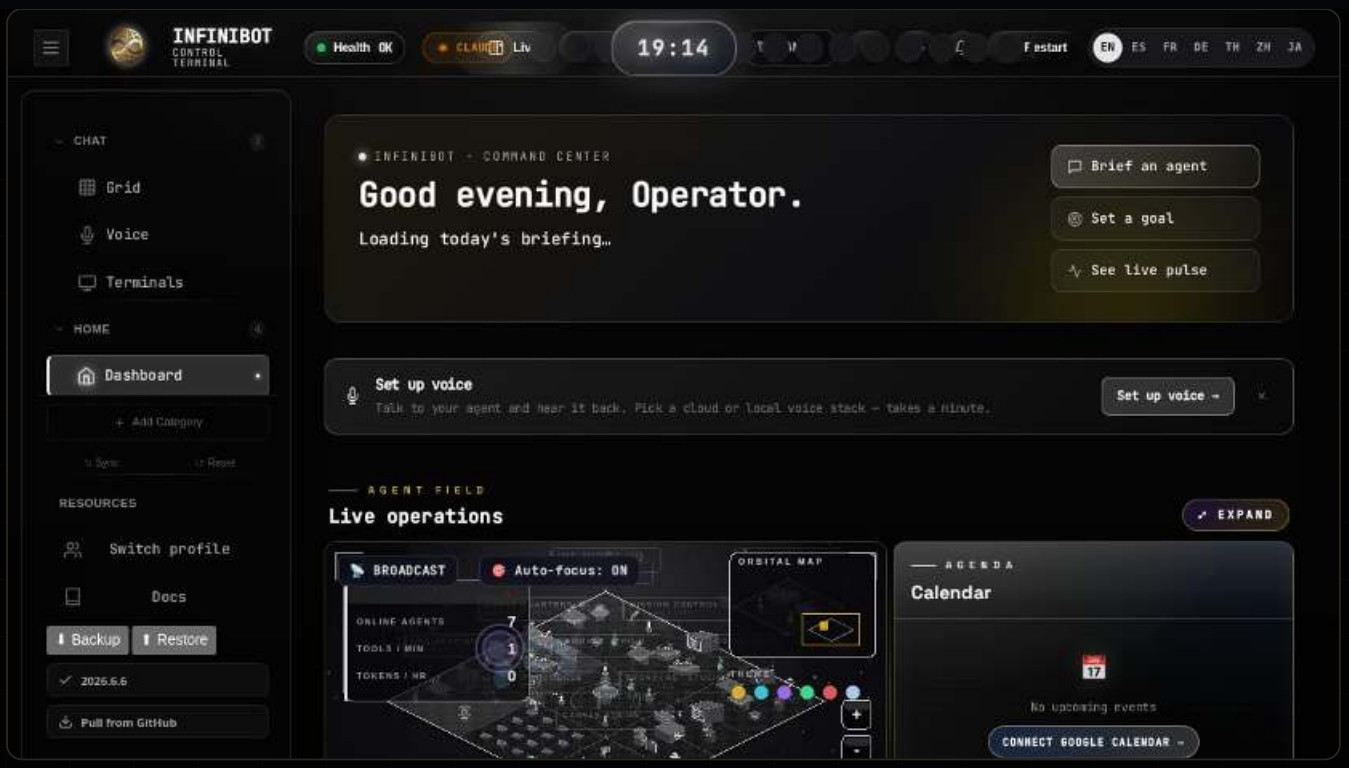


Figure · Federation at scale — one master hub, many satellites; spoke-to-spoke traffic tunnels through the master.

> You've made money with one InfiniBot on one machine. This chapter is about multiplying that: more people sharing one gateway, more machines working as one mesh, keeping spend visible across providers, and turning the whole thing into a product other people pay you for.



A FLEET OF AGENTS WORKING IN PARALLEL ACROSS THE MESH

Scaling InfiniBot happens on four axes. You can pull any one of them without the others, but together they compound:

1. **Multi-user profiles** — more *people* on one gateway, isolated from each other.
2. **Federation** — more *machines* acting as one fleet.
3. **Plan-aware credits** — *visibility and control* over what every provider is costing you.
4. **Monetization** — turning the operation into *recurring revenue* via Whop + KinglyVault.

1. Multi-user profiles: many people, one gateway

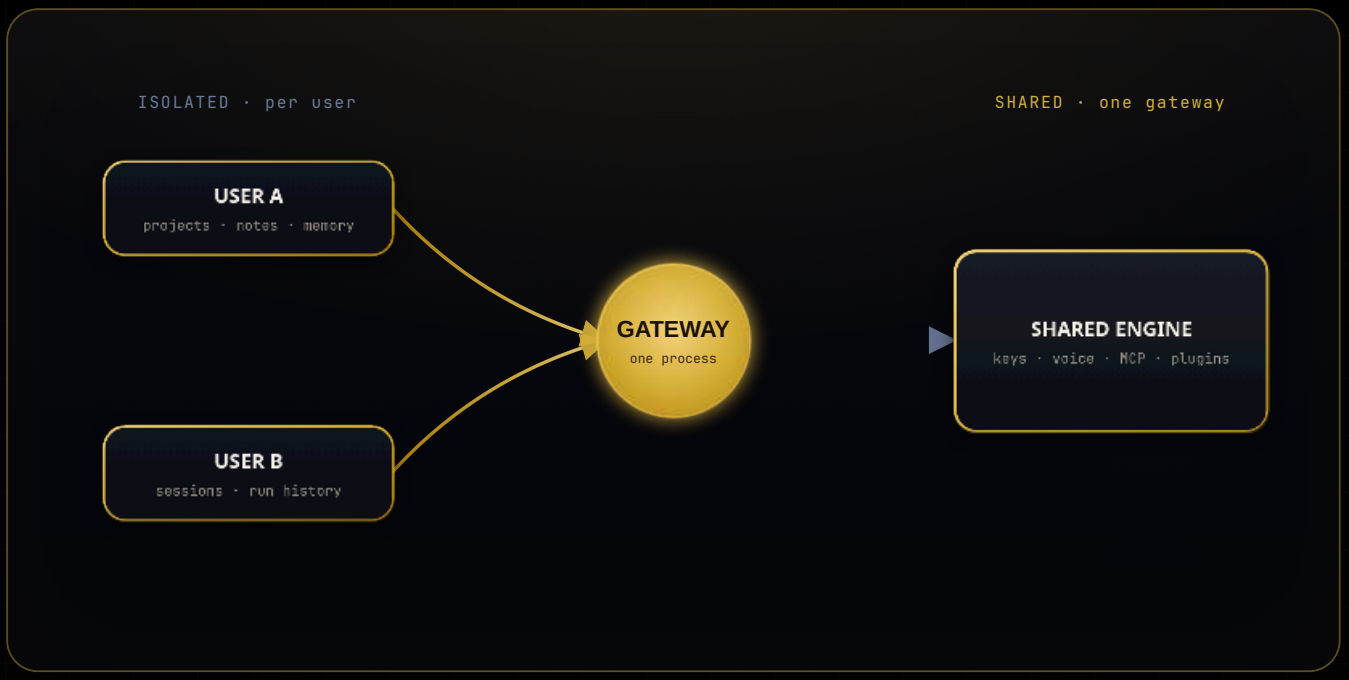


Figure · The isolation boundary — each user owns their work; everyone shares the engine.

ISOLATED · PER USER

- Projects & tasks (the kanban board)
- Notes
- Agent memory
- Sessions & run history

SHARED · ACROSS THE GATEWAY

- The gateway process itself
- Model / API providers (BYOK keys)
- Voice configuration
- Federation links
- MCP servers & connected tools
- Stored credentials
- Installed plugins

> UNDER THE HOOD
>
> Isolation is namespaced per user via **resolveStateDir()**, gated by a profile picker. Runtime DBs live in **~/.infinibot** — a gateway restart guarantees a fully clean switch.

A single InfiniBot gateway can host **multiple users**, each with their own isolated workspace, without any of them needing a password to *use* the machine. Profiles provide **password-free per-user data isolation**: each user gets a namespaced state directory so their projects, tasks, notes, agent memory, and sessions are theirs alone.

What is isolated vs. shared

This distinction is the whole game — get it wrong and you'll either leak data between users or duplicate infrastructure you didn't need to.

Per-user (isolated):

- Projects and tasks (the kanban board)
- Notes
- Agent memory
- Sessions and run history

Shared across all users on the gateway:

- The gateway process itself
- Model/API providers (your BYOK keys)
- Voice configuration
- Federation links
- MCP servers and connected tools
- Stored credentials
- Installed plugins

In practice: everyone on the gateway shares the *engine* (keys, voice, tools) but owns their *work* (projects, notes, memory). That's the right default — you pay for one set of provider keys, and each operator gets a private workspace.

How it works under the hood

Isolation is implemented by namespacing the state directory per user (`resolveStateDir()`), gated behind a profile picker screen at launch. One caveat worth knowing: switching the active user mid-session can leave cached singletons pointing at the previous user's data — a **gateway restart** guarantees a fully clean switch. For day-to-day use, log in as your user from the picker and you're isolated.

> **Where state actually lives.** Runtime databases live in `~/.infinibot` , *not* in the repo's `state/` directory. If you're hunting for "where did my projects go," that's the folder.

When to reach for profiles

- A small team sharing one beefy machine.
- A household or studio where each person wants their own agents and notes.
- A consultancy where each client engagement gets its own profile so work never bleeds across accounts.

2. Federation: many machines, one fleet

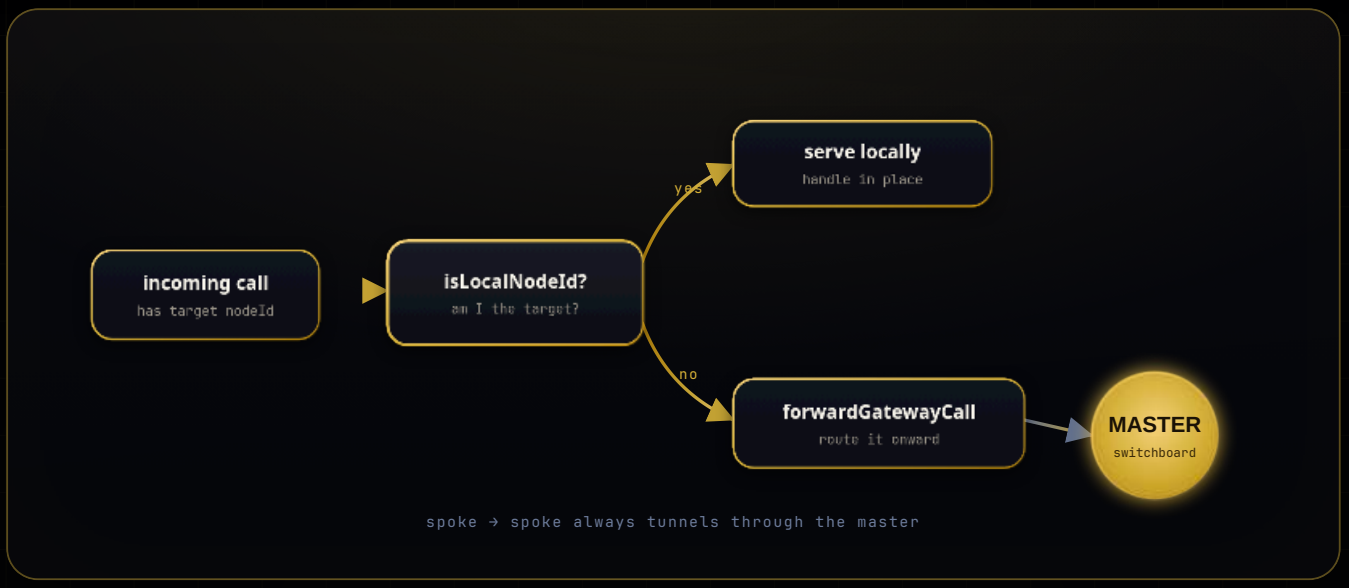
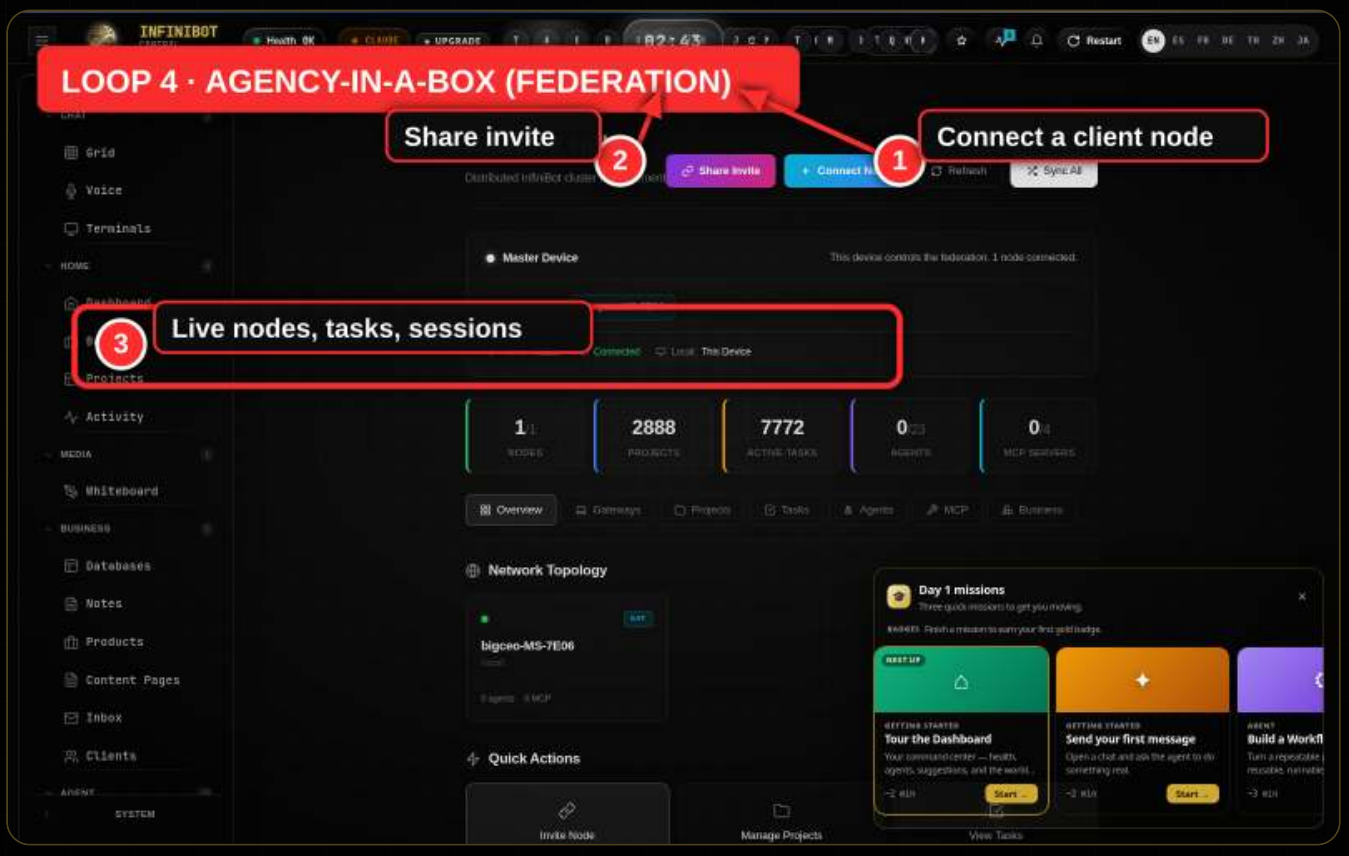
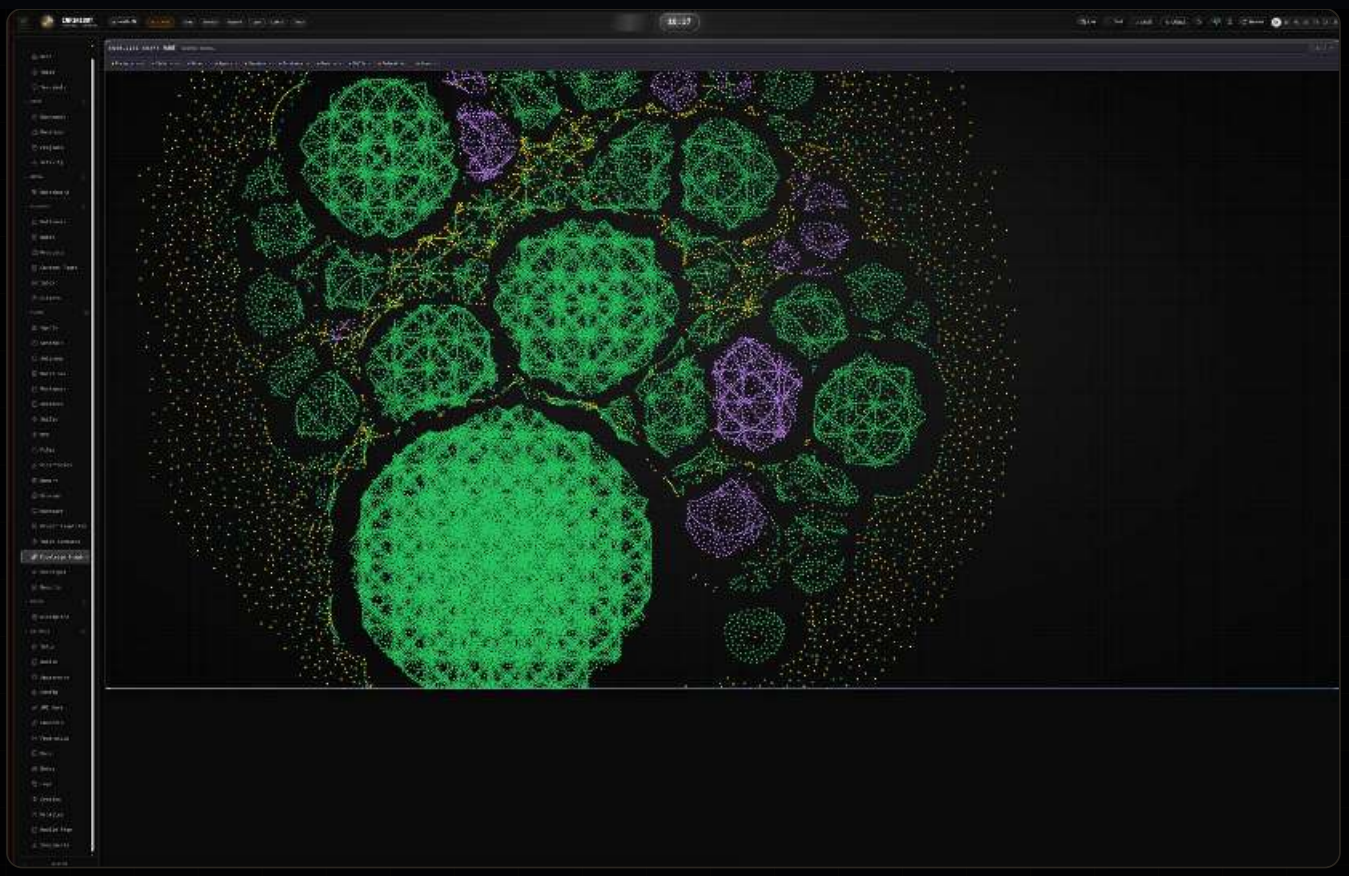


Figure · Routing — local calls are served in place; everything else is forwarded, and spokes meet only at the master.



FEDERATION IN PRACTICE — AGENTS AND SESSIONS REACHABLE ACROSS THE WHOLE FLEET



AGENTS DISTRIBUTED ACROSS FEDERATED NODES

Federation lets **multiple InfiniBot machines act as a single fleet.** You designate one node as the **master** (the hub) and connect **satellite** nodes (the spokes). Agents, sessions, and tools become reachable across the whole mesh.

Hub-and-spoke, not a full mesh

The critical mental model: federation routing is **hub-and-spoke**, not peer-to-peer.

- Every satellite connects to the **master**.
- Satellite-to-satellite traffic is **tunneled through the master** — two spokes never talk directly.
- The two choke points in the code are `isLocalNodeId` (am I the target?) and `forwardGatewayCall` (route it onward). Everything that crosses a node boundary passes one of these.

This is why the master must stay up: it's the switchboard. If a satellite drops, the rest of the fleet is fine; if the master drops, cross-node calls pause until it's back.

Routing a call to the right node

Gateway methods and UI actions target a node by `nodeId` and invoke with `hub.invokeOnNode`. When you spawn an agent, you can pin it to a specific node (or let the fleet place it by capability — GPU VRAM, installed tools, etc.). A few facts that save hours:

- **New gateway methods need a restart to be served** — the registry is built at boot. Add a method, restart the node that serves it.
- **Federated sub-agents are placed by `nodeId`, not node name.** A spawn that lands on the wrong "island" in the world map usually means something read the local session registry instead of the remote one.

- **Self-test before you trust it:** run `node scripts/federation-selftest.mjs` on the master to confirm round-trip routing.

One-command fleet sync

To bring a new machine into the fleet, sign in to **KinglyVault → /infinibot** and grab the satellite-setup script for your platform: `satellite-setup.sh` (Linux/macOS) or `satellite-setup.ps1` (Windows). Run it on the new machine — it installs InfiniBot, configures it, and registers the satellite against your master automatically. **Install outside any OneDrive-synced path** — file-locking and partial syncs cause intermittent breakage.

When to federate

- You have a spare machine (or a cheap cloud box) and want it pulling its weight.
- You need GPU on one node (local voice/models) and cheap always-on compute on another.
- You're running client work that should be physically separated onto a dedicated box.

3. Plan-aware credits: see and control the spend

24h	2	1	$0
ROLLING USAGE TIME SERIES	PLAN MODES · AUTO / MANUAL	CONFIG · PROVIDER-PLANS.JSON	RESALE MARKUP — PURE BYOK

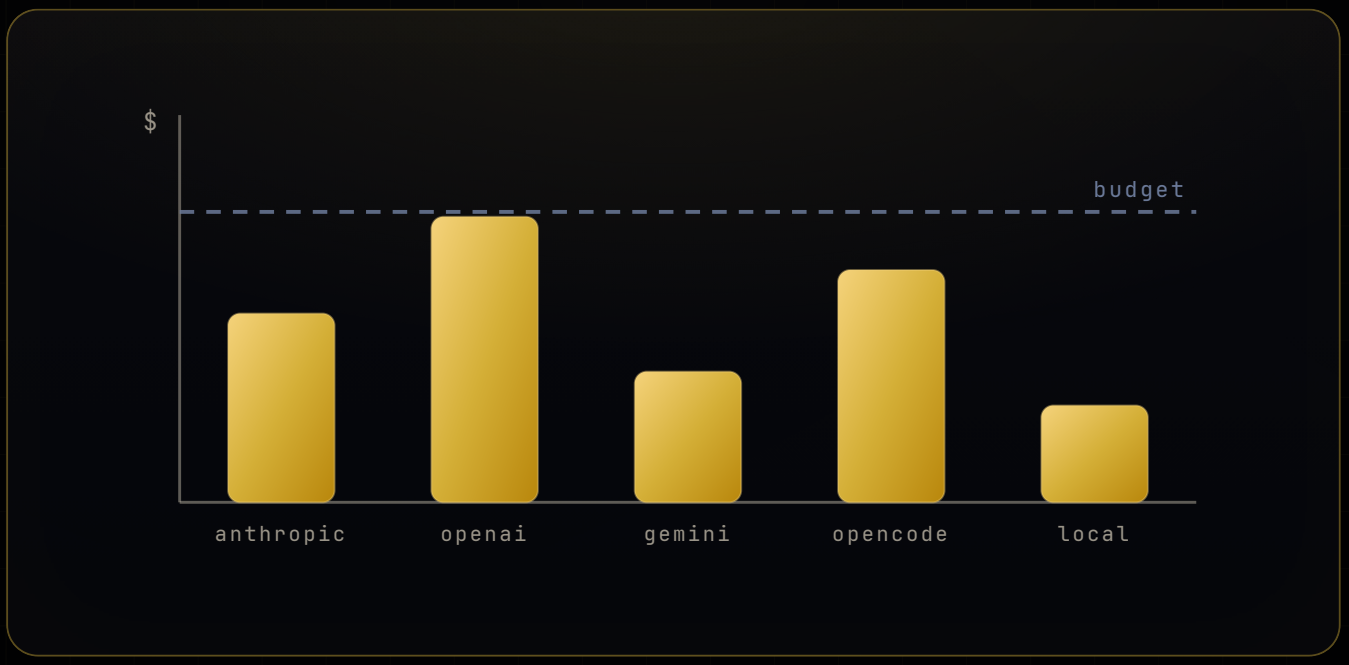
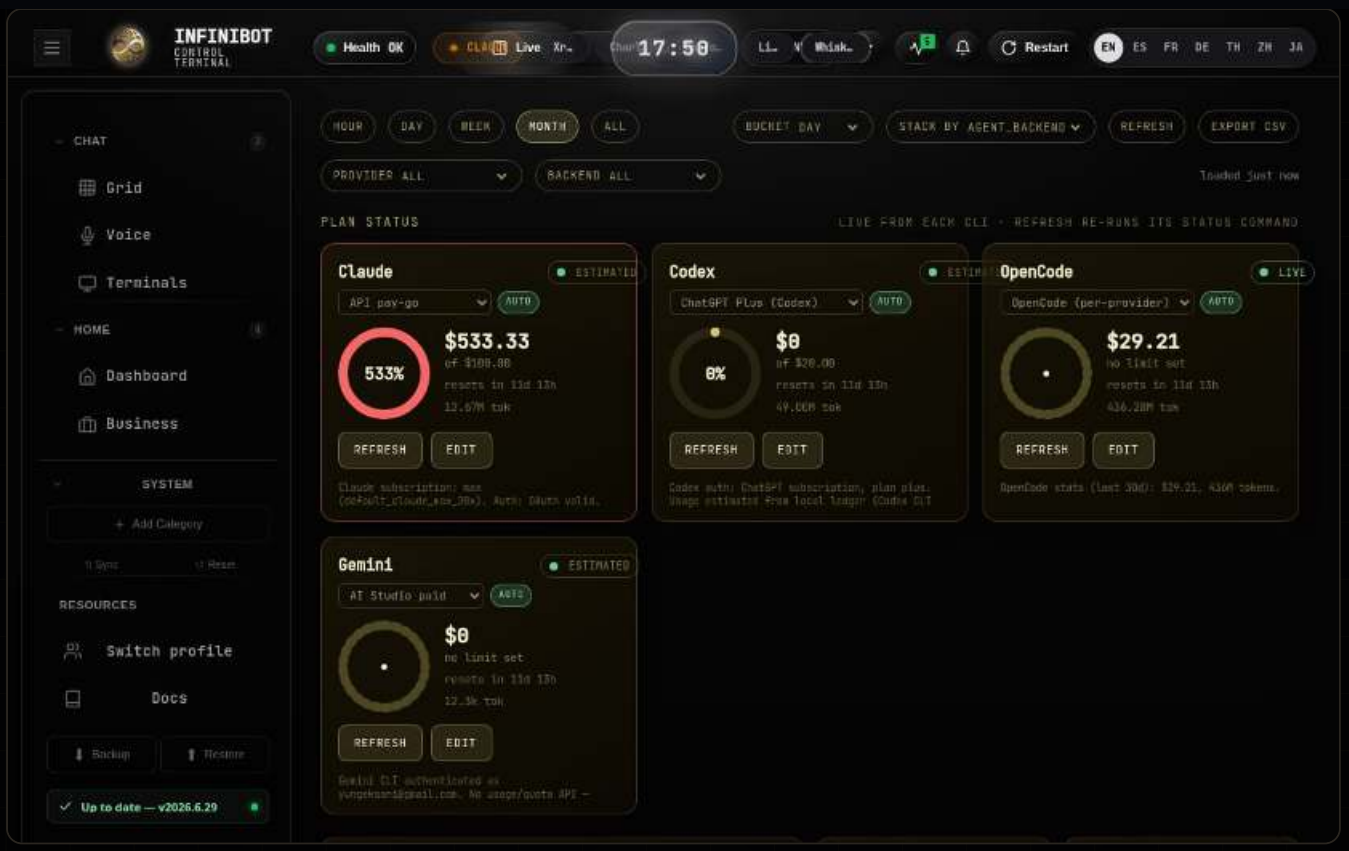


Figure · Spend at a glance — daily cost by provider against your budget line.



THE CREDIT USAGE PAGE — SPEND ACROSS PROVIDERS AND AGENTS

Because InfiniBot is BYOK, *you* pay your providers directly — which means you need to see that spend clearly. The **plan-aware credits** system tracks provider plans and quotas, pulls live usage, and surfaces it in the **Credit Usage** page.

What it tracks

- **Provider plans and quotas** — registry + pullers + a refresh loop, configured in `~/.infinibot/provider-plans.json`.
- **Live usage** derived from `costUsd`, grouped by agent/backend, available as a 24-hour time series.
- **CLI-backed usage** for coder backends (e.g. OpenCode stats), so terminal agents show up alongside API usage.

Plan selection: auto vs. manual

The credits page lets you pick your provider plan, or let InfiniBot **auto-detect** it. There's an `auto` / `manual` mode flag per provider in `provider-plans.json`:

- **Auto** suggests a `detectedPlan` from probing the provider and follows it.
- **Manual** pins the plan you chose and *won't* be silently overridden by detection.

Use manual when you know your exact plan and don't want a probe second-guessing it; use auto when you'd rather it just figure it out. The page shows a "use detected" prompt when auto and manual disagree.

> **Restart note.** Credits *backend* changes (new RPCs, plan registry tweaks) take effect after a gateway restart; the *UI* updates live after a build. If a new credits feature "isn't there," you're likely looking at a running gateway from before the change.

Why this matters for scaling

When you go from one agent to ten, or one machine to five, spend can creep without you noticing. The credits page is your cost dashboard — check it before you scale a workflow 10×, not after the bill arrives.

4. Monetization: turn the operation into revenue

Figure · Monetization ladder — services → access → knowledge, all on a BYOK foundation.

◆ **Proof in use beats feature demos.** The strongest InfiniBot marketing is real production usage — the fleet doing actual client work, real revenue, real screenshots. People buy the operation they can see running.

SELLING ACCESS — THE PRODUCTIZED FUNNEL

InfiniBot's own business model is also *your* template for productizing it: **KinglyVault + Whop + BYOK.**

How InfiniBot monetizes (and how you can too)

- **KinglyVault issues Whop licenses.** Access is gated behind a Whop-issued license checked at a login gate. KinglyVault is the content/license hub; Whop is the storefront and payment rail.
- **Users BYOK their own LLM keys.** Because every buyer brings their own provider key, there is **no token/inference resale** and **zero per-user inference overhead.** That's pure software margin — you sell software and education, not compute.

This is the cleanest SaaS-style model available to a solo operator: recurring revenue through Whop, fulfillment through KinglyVault content, and no variable cost-of-goods because the customer pays their own AI provider.

The pieces and where they live

- **Landing page:** `docs/landing.html` (the public sales page).
- **Whop integration + credentials:** in the KinglyVault repo (canonical path `~/Desktop/KinglyVault-main` — note the `KK/KinglyVault-main` copy is a *stale fork*, never ship from it).
- **Pricing:** `infinibot-saas/PRICING.md` plus `src/credits/pricing.*`.
- **Upgrade flow:** the landing page's `/upgrade` redirects to the Vault `infinibot-upgrade` page.

Your monetization ladder

1. **Sell outcomes (services).** Use the CRM + content workflows from Part 3 to deliver client work. Lowest setup, fastest cash.
2. **Sell access (productized).** Package a workflow or a configured InfiniBot as a Whop product. Recurring revenue, BYOK keeps your costs flat.
3. **Sell knowledge (education).** Bundle playbooks and courses through KinglyVault. The thing you're reading is itself an example of this tier.

> **Positioning tip — proof in use.** The strongest InfiniBot marketing is *real production usage*, not feature demos. Show the fleet doing actual client work, real revenue, real screenshots. People buy the operation they can see running, not the feature list.

Putting it together

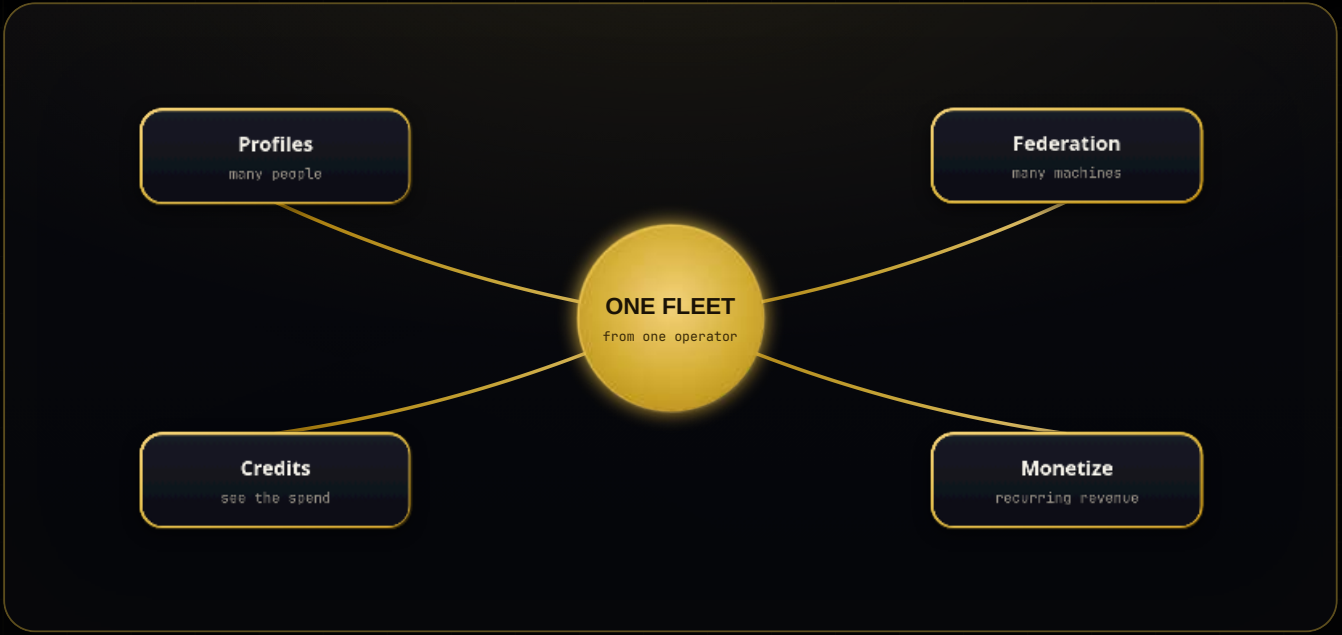


Figure · Four axes of scale compounding into one fleet, run by one operator.

A mature InfiniBot operation looks like this: a **master** node on reliable hardware running shared keys and voice; one or two **satellites** for GPU or isolation; **profiles** for each operator or client; the **credits page** open as a cost dashboard; and a **Whop product** turning the whole thing into recurring revenue while every customer brings their own key.

You don't build that on day one. You build it one axis at a time — and each axis you add makes the others worth more.

Next: the Glossary & Troubleshooting chapter — every term defined, and the top 20 things that go wrong, with fixes.

TRY THIS IN INFINIBOT NEXT CHAPTER →

06

PART 06 · REFERENCE

Glossary & Troubleshooting

> Every term, every trap, every fix — the reference you reach for when something breaks.

REFERENCE

Glossary & Troubleshooting

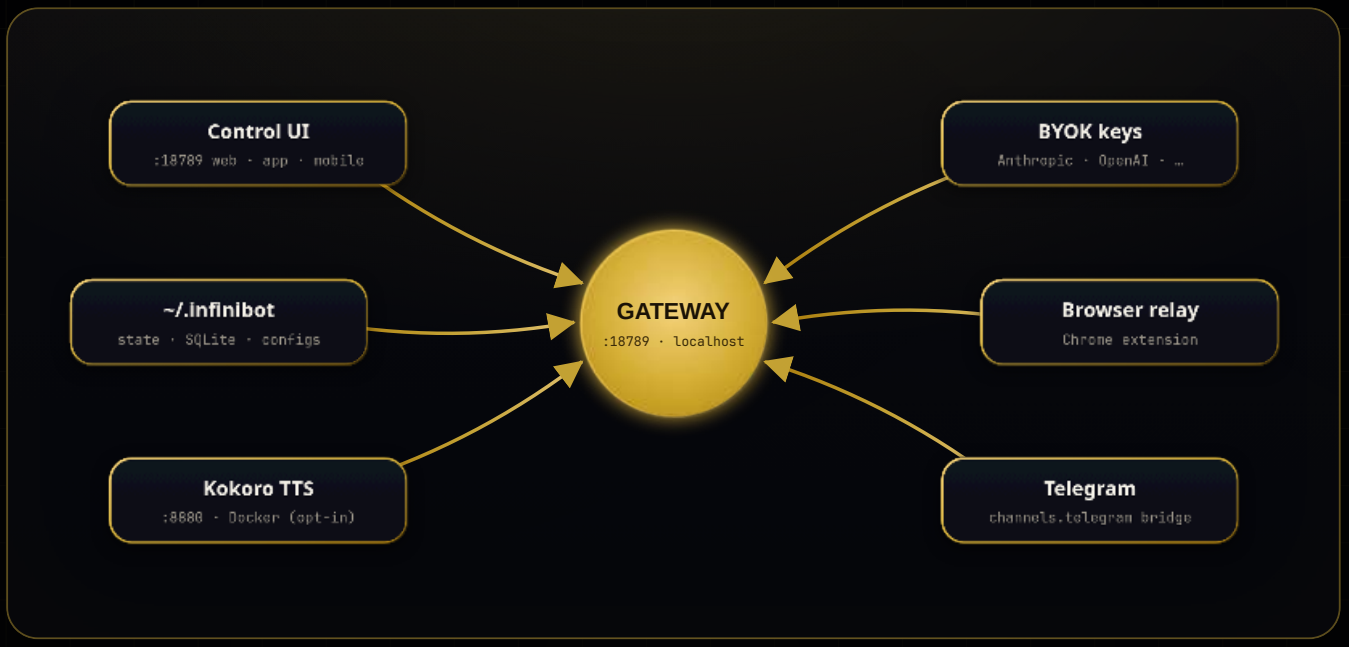


Figure · Service & port map — the gateway and the services around it.

> The reference you flip to when a word doesn't make sense or something just broke. The glossary defines every term used in this book. The troubleshooting section is the **top 20 beginner traps**, each with the symptom, the cause, and the fix. When InfiniBot misbehaves, scan here *first* — the odds are high it's one of these.

Glossary

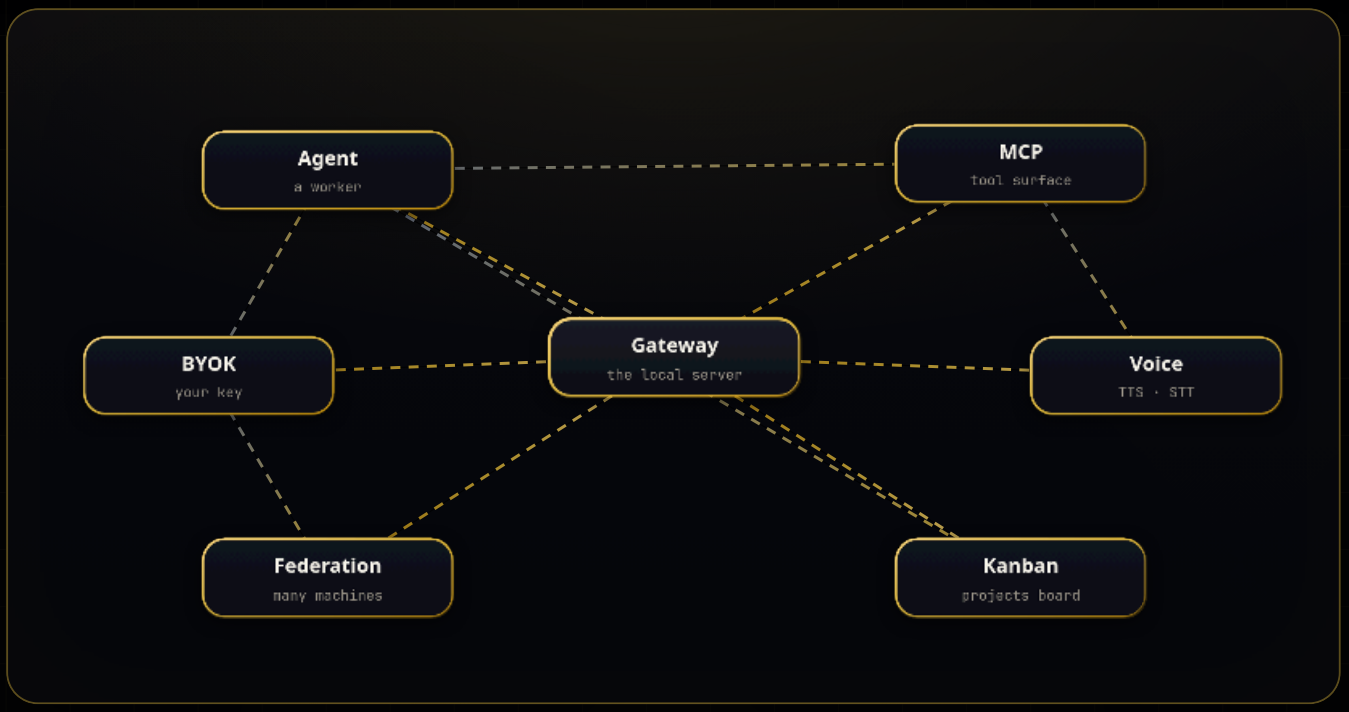


Figure · Term constellation — how the core concepts relate.

Agent. An autonomous worker spawned by the gateway that performs a task — writing code, drafting content, running research, answering a channel. Agents have their own session, memory, and run history.

BYOK (Bring Your Own Key). InfiniBot's core model: you supply your own model-provider API key. InfiniBot never resells tokens or inference; your provider bills you directly at cost. This is why scaling is a software decision, not a billing event.

Backend (coder backend). The CLI/engine an agent uses to write code: Claude Code, Codex, OpenCode, or Gemini CLI. Selectable per agent; detection of installed backends is PATH-sensitive (see Trap #13).

Control UI. The browser/desktop/mobile interface you drive InfiniBot from. Served by the gateway at `http://localhost:18789` with path-based routes (`/dashboard` , `/grid` , `/chat` , `/projects` , `/credits` , etc.).

Federation. Linking multiple InfiniBot machines into one fleet. **Hub-and-spoke:** satellites connect to a master; satellite-to-satellite traffic tunnels through the master.

Gateway. The local-first server process at the heart of InfiniBot. Hosts agents, serves the control UI, holds shared keys/voice/tools. Runs on *your* machine.

Kanban. The projects board. Projects contain tasks; the `/projects` route renders the kanban view. Note: changing a project's status cascades to all its child tasks.

Kokoro. A local neural TTS engine, run as a Docker container (`infinibot-kokoro` , port 8880). One of the opt-in local voice options; *not* auto-started by the gateway (see Trap #3).

KinglyVault. InfiniBot's content/license hub. Issues Whop licenses and hosts educational content. Canonical repo: `~/Desktop/KinglyVault-main` .

Master / Satellite. In federation, the **master** is the hub all satellites connect to; **satellites** are the spokes. The master is the switchboard for cross-node calls.

MCP (Model Context Protocol). The protocol by which InfiniBot exposes tools to agents. The external MCP tool surface is *dynamic* — derived at runtime, not from a static list.

Mesh / Fleet. The collection of agents (and, with federation, machines) working together.

Node. A single InfiniBot machine in a federation, identified by `nodeId`.

Profile (multi-user). A namespaced, isolated workspace for one user on a shared gateway. Per-user: projects, tasks, notes, agent memory, sessions. Shared: gateway, keys, voice, federation, MCP, credentials, plugins.

RPC. A gateway method callable by the UI/agents. **New RPCs require a gateway restart** to be served (the registry builds at boot).

State directory. Where runtime databases live: `~/.infinibot` — *not* the repo's `state/` folder.

TTS / STT. Text-to-speech / speech-to-text. Cloud by default (zero install, every OS); local engines (Piper, Whisper, Kokoro) are opt-in.

Whop. The storefront/payment rail InfiniBot uses for licensing and recurring revenue. (Heard as "WAP" in voice transcripts.)

flypic. InfiniBot's image/visual generation pipeline. (Heard as "flypig" in voice transcripts.)

Troubleshooting: the top 20 traps

↻ IT NEEDS A RESTART

- ✓ **#5** New RPC isn't there → restart; the registry builds at boot.
- ✓ **#7** Switched users, old data → restart for a clean profile switch.
- ✓ **#12** Voice "saved" but ignored → restart so `voice.json` reloads.
- ✓ **#16** Federation calls hang → master must be up; restart serving node.
- ✓ **#19** Telegram won't send → set `channels.telegram`, restart.

▦ IT NEEDS A BUILD

- ✓ **#15** Lit/UI change invisible → run `vite build` (→ dist/control-ui).
- ✓ Backend RPC change → restart instead. Many features need **both**.
- ✓ Dismiss the profile picker first, or the route may not mount.

⟲ STALE DIST / DAEMON RESET

- ✓ **#8** "Missing" MCP tool → stale `dist/` ; verify via mcp-stdio.js.
- ✓ **#9** Edits vanished → auto-rebuild daemon reset the tree; keep a /tmp patch.
- ✓ **#10** `pnpm build` broke things → use `tsc` ; clear .vite cache.

⌖ POINTED AT THE WRONG THING

- ✓ **#1** UI won't load → use **:18789**, not 47913.
- ✓ **#6** Data missing → it lives in `~/.infinibot` , not repo `state/` .
- ✓ **#11** MCP note not in UI → use `ui_control` (split stores).
- ✓ **#17** Flaky on OneDrive → clone outside synced paths.
- ✓ **#18** Vault change ignored → ship from canonical KinglyVault-main.

⚙ SETUP STEP SKIPPED

- ✓ **#2** Agents do nothing → no BYOK key; paste one in the wizard.
- ✓ **#3** Voice silent → Kokoro Docker isn't running; `docker ps` or use cloud.
- ✓ **#4** Browser does nothing → load the relay extension once, manually.

⤫ DETECTION & ROUTING QUIRKS

- ✓ **#13** Backend "not installed" → stripped PATH; Refresh or set `binPath` .
- ✓ **#14** Mid-run correction ignored → it hits a secretary; respawn instead.
- ✓ **#20** Plan keeps reverting → switch the provider to **manual** mode.

> REFLEX
>
> Most failures reduce to four checks — **restart**, **build**, **stale/reset**, or **wrong target**. Run them in that order and you'll self-diagnose 90% of what goes wrong.

1. The UI won't load — wrong port (18789 vs 47913)

Symptom: Browser can't connect, or the mobile app says "voice won't connect." **Cause:** The gateway serves WebSocket/UI on `:18789` . Port `47913` is a *canonical-but-unused* federation port — easy to confuse. A past mobile default pointed at `ws://…:47913` and silently failed. **Fix:** Use `http://localhost:18789` for the UI and `ws://<host-ip>:18789` for the gateway WebSocket. On mobile, set the gateway URL to your host's LAN IP on **18789**, not 47913.

2. Agents do nothing / "no model" — BYOK key missing

Symptom: Agents spawn but never produce output, or error about no provider. **Cause:** No model-provider API key configured. InfiniBot is BYOK; with no key there's no intelligence to call. **Fix:** Open the first-run wizard (or

Settings → Providers) and paste **at least one** provider key (Anthropic recommended). Confirm it on the Credits page once usage shows up.

3. Voice goes silent — Kokoro Docker not running

Symptom: Local TTS produces no audio; you'd selected Kokoro. **Cause:** Kokoro runs as the `infinibot-kokoro` Docker container on port **8880** (from `docker-compose.voice.yaml`). The gateway does **not** auto-start it. **Fix:** Run `docker ps` to confirm the container is up. If not, start it (`docker compose -f docker-compose.voice.yaml up -d`). Or switch back to **cloud voice**, which needs no Docker and is the default. Don't blame the code before checking `docker ps`.

4. Browser automation does nothing — Chrome relay extension not installed

Symptom: Browser-driving agents can't attach; the relay seems dead. **Cause:** On Chrome 137+ the `--load-extension` flag is **silently ignored** on stable channel. The relay extension never loads. **Fix:** Install it **once, manually**: `chrome://extensions` → enable **Developer mode** → **Load unpacked** → point at the relay extension folder. After that, spawned agents auto-attach.

5. "I changed code but nothing changed" — new RPC needs a restart

Symptom: A new feature/RPC you added isn't available in the running UI. **Cause:** The gateway's RPC registry is built at boot. New gateway methods aren't served until restart. **Fix:** Restart the gateway. (UI-only changes are different — those go live after a build; see Trap #15.)

6. "Where did my projects/notes go?" — wrong state directory

Symptom: Data seems missing; you're looking in the repo's `state/`. **Cause:** Runtime databases live in `~/.infinibot`, not the repo `state/` folder. **Fix:** Look in `~/.infinibot`. That's where the SQLite databases, configs (`infinibot.json`, `voice.json`, `provider-plans.json`), and per-user namespaced dirs live.

7. Switched users but still see old data — cached singletons

Symptom: After switching profiles, you still see the previous user's projects/notes. **Cause:** Some singletons are cached at boot and don't fully re-resolve the per-user state dir on a live switch. **Fix: Restart the gateway** for a guaranteed-clean user switch. For normal use, pick your user at the profile picker on launch.

8. A "missing" MCP tool that should exist — stale dist

Symptom: A tool you know exists isn't in the MCP surface; grepping `src/mcp` for its name finds nothing. **Cause:** The external MCP tool surface is *dynamically derived* (`agent.tools.list → createInfiniBotTools`) — there's no static list to grep. A genuinely "missing" tool is almost always a **stale** `dist/` from the rebuild daemon mid-cycle. **Fix:** Rebuild, then verify against ground truth by driving `dist/cli/mcp-stdio.js`. Don't trust a `src/mcp` grep for tool presence.

9. Edits vanished mid-session — the auto-rebuild daemon reset the tree

Symptom: Code you just wrote disappeared. **Cause:** An auto-rebuild/auto-park daemon periodically **git-resets the working tree** (parks → reset → restores), and can drop edits made mid-cycle. **Fix:** Keep a `/tmp` patch backup of in-progress work (`git diff > /tmp/wip.patch`) so you can re-apply if a cycle eats it. Commit early on a branch.

10. `pnpm build` "broke everything" — use `tsc`, and beware clean

Symptom: After a build, `dist` is empty or a phantom error appears. **Cause:** A full `pnpm build` can run a clean that **wipes** `dist`; a stale `ui/node_modules/.vite` cache can also throw phantom errors. **Fix:** For a quick type-check/compile use `tsc`. If you hit phantom build errors, clear `ui/node_modules/.vite`. Reserve full `pnpm build` for when you actually need a clean rebuild.

11. Notes created via MCP don't show in the UI — split stores

Symptom: You created a note through an MCP tool, but the Notes view doesn't show it. **Cause:** `notes.create` (MCP) writes to a **different store** than the UI Notes view reads. **Fix:** Use `ui_control dispatch view=notes op=create` so notes land in the store the UI reads. (And target a single tab — a dispatch with no tab target writes to *every* open tab and creates duplicates.)

12. Voice setup "saved" but runtime ignores it — dual config files

Symptom: You configured voice via CLI/onboard, but the runtime still uses the old voice. **Cause:** Voice has **two stores**: rich config in `infinibot.json::voice` vs. the runtime `voice.json`. The runtime reads **only** `voice.json`, so a CLI onboard that wrote the wrong file was a silent no-op. (Native Kokoro also needs `voice-settings.json`.) **Fix:** Use the in-app voice setup, which now bridges both files. After changes, restart the gateway so the runtime picks up `voice.json`.

13. Coder backend shows "not installed" but it is — stripped PATH

Symptom: The setup page says a backend (Claude/Codex/etc.) isn't installed, though it runs fine in your terminal. **Cause:** The gateway daemon ran detection with a **stripped PATH** (bare `execFileSync`), so it couldn't find the binary. **Fix:** This is handled by `bin-detect.ts` (enriched PATH). If it still shows wrong, use the manual **Refresh** on the setup page, or set the explicit `binPath`. The page also refreshes on focus.

14. Mid-run "corrections" to a coder agent are ignored — routing trap

Symptom: You `sessions_send` a correction to a running coder agent; it acknowledges but the code never changes. **Cause:** A mid-run message to a `claude_coder`'s child session lands in an **Ollama secretary** that just acknowledges — the real coder process never sees it. **Fix:** Get the brief right the first time, or **respawn** with a corrected brief. Don't rely on mid-run `sessions_send` to steer a coder.

15. UI change not visible — build vs. restart confusion

Symptom: You changed a Lit view; the running UI looks the same. **Cause:** UI changes need a **Vite build** (→ `dist/control-ui`, served live), which is *different* from a gateway restart. Conversely, backend RPC changes need a restart, not a build. **Fix:** UI → `vite build`. Backend RPC → restart. Many features need *both*. Also dismiss the profile picker before navigating to a route, or the route may not mount.

16. Federation calls hang or land on the wrong node

Symptom: Cross-node calls time out, or a federated agent appears on the wrong "island." **Cause:** Federation is hub-and-spoke; the **master must be up** to relay satellite-to-satellite calls. Misplacement usually means something read the *local* session registry instead of the *remote* one — placement is by `nodeId`, not node name. **Fix:** Confirm the master is running. Run `node scripts/federation-selftest.mjs` on the master to verify round-trip routing. Remember new gateway methods need a restart on the serving node.

17. Installed to a OneDrive path — intermittent breakage

Symptom: Random file-lock errors, partial files, flaky builds. **Cause:** OneDrive-synced folders interfere with file locking. **Fix:** Install InfiniBot **outside** OneDrive-synced paths (e.g. `C:\InfiniBot`, not inside `Documents` or

`Desktop` if those sync to OneDrive). Download the latest bundle or satellite-setup script from your **KinglyVault → /infinibot** member page.

18. Shipped from the wrong KinglyVault — stale fork

Symptom: Vault content/license changes don't take effect. **Cause:** Two copies exist; `~/Desktop/Development/KK/KinglyVault-main` is a **stale fork**. **Fix:** Use the canonical `~/Desktop/KinglyVault-main`. Never dispatch agents at, or ship from, the `KK/` copy.

19. Telegram bridge won't send — wrong config source

Symptom: Reports/alerts don't reach Telegram. **Cause:** The real bot credentials live in `channels.telegram` inside `~/.infinibot/infinibot.json` (`botToken` + `defaultChatId`). The discipline-store `telegramChatId` path is disabled/empty. **Fix:** Set `channels.telegram.botToken` and `defaultChatId` in `~/.infinibot/infinibot.json`, then restart.

20. Plan selection keeps reverting — auto detection overriding you

Symptom: You pick a provider plan on the Credits page; it snaps back to something else. **Cause:** Auto-detection (`probe.plan`) silently overrode your manual choice. **Fix:** Switch that provider to **manual** mode in `~/.infinibot/provider-plans.json` (or via the inline picker's AUTO toggle). Manual pins your selection; auto follows the detected plan and shows a "use detected" prompt instead of overriding.

Still stuck?

Most issues reduce to one of four root causes. When in doubt, check them in order:

1. **Did a new backend feature need a restart?** (Traps #5, #7, #12, #16, #19)
2. **Did a UI change need a build?** (Trap #15)
3. **Is the dist stale, or did a daemon reset the tree?** (Traps #8, #9, #10)
4. **Am I pointed at the right thing** — port 18789, `~/.infinibot` , canonical repo, the right config file? (Traps #1, #6, #11, #17, #18)

Nail those four reflexes and you'll self-diagnose 90% of what goes wrong.

TRY THIS IN INFINIBOT JOIN THE COMMUNITY