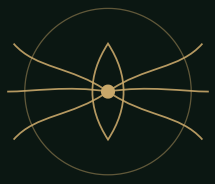

SCOUT

The Agent That Hunts Jobs While You Sleep

How I wired one AI agent to LinkedIn, a database, and a scheduler — so it finds real roles, scores every one against me, and drafts the outreach on its own. The whole architecture, generalized so you can build it on your stack this weekend.

BY **HUNTER GRANT** · HUNTERGRANT.DEV

BUILT WITH CLAUDE — FABLE 5
DAY 8 OF BUILDING IN PUBLIC

START HERE

A tracker isn't a list. It's a *loop*.

Most people job-hunt by opening LinkedIn and scrolling. You search, you skim, you bookmark a few, you close the tab. That's a chore — and it depends on you showing up every day to do it.

An agent flips it. You give one model a place to keep work, a heartbeat to wake it, a way to see the market, and a definition of good — and it hunts on its own. It pulls fresh roles, scores each one against you, files the keepers, and writes the first outreach, while you're asleep or at the gym.

> *The job board doesn't wait for you to look. It fills itself overnight.*

There's no secret model and no magic prompt — just four moving parts wired into one loop. Everything here is generalized: swap in your own stack and the shape holds. The tools I name are the ones I actually use.

THE FOUR — AND THE LOOP THAT JOINS THEM

01	The Spine	A DATABASE THAT IS QUEUE AND MEMORY
02	The Heartbeat	A CRON THAT WAKES THE AGENT
03	The Eyes	LINKEDIN, AND THE ASYNC CATCH
04	The Judgment	SCORE EVERY ROLE AGAINST YOU
+	The Reach	DRAFT OUTREACH TO A REVIEW QUEUE

01 The Spine

A database that is both queue and memory.

USE IT WHEN

Before anything else. The agent needs somewhere durable to take work from and put results into — so a crash, a restart, or a missed night never loses the thread.

SCHEMA · SUPABASE

```
-- two tables carry the whole system.

-- 1) the work queue: every job the agent will do, one row each.
create table tasks (
  id          uuid primary key default gen_random_uuid(),
  kind        text not null,            -- find_jobs | collect |
research
  status      text default 'pending',   -- pending|running|done|failed
  payload     jsonb default '{}',       -- e.g. {keyword, location}
  not_before  timestamptz default now(),-- delay without a timer
  attempts    int default 0,
  idem_key    text unique               -- never do the same job twice
);

-- 2) the job board: every role it finds, scored and deduped.
create table roles (
  id        uuid primary key default gen_random_uuid(),
  company text, title text, jd_url text, location text,
  fit_score int,                      -- 0-100, from part 04
  notes jsonb,                        -- gaps + why it fits
  status text default 'researching',
  unique (company, title)          -- the dedupe key
);
```

> **Why it works.** An agent without durable state is a goldfish — every run starts from nothing. Putting the queue and the results in one database means the loop is idempotent and resumable: it can die mid-hunt and pick up exactly where it left off. The *idem_key* is the quiet hero — it guarantees the same role never gets scored, or the same search never gets run, twice.

02 The Heartbeat

A cron that wakes the agent — so you don't have to.

USE IT WHEN

The moment you want it running without you. This is the single line between a tool you open and a system that shows up on its own.

WORKFLOW · N8N

```
-- two nodes. that's the whole scheduler.

[ Schedule Trigger ]  every 30 minutes
        |
        v
[ HTTP Request ]      POST https://your-app/api/tick
                       header: Authorization: Bearer <secret>

-- /api/tick is one endpoint that does one thing:
--   1. claim the next due task (status=pending, not_before <= now)
--   2. run it                  (route on task.kind)
--   3. write the result back, mark it done, exit

-- keep it boring: no long jobs inside the tick. if a step
-- needs to wait, it queues a follow-up task and returns.
```

> **Why it works.** The cron is dumb on purpose — it just pokes one endpoint on a cadence. All the intelligence lives behind */tick*, which you can also hit by hand to test. Because each tick claims one due task and returns fast, the system can't pile up or run away: work that isn't ready simply isn't claimed. A heartbeat, not a marathon.

03 The Eyes

LinkedIn — and the async catch that bites everyone.

USE IT WHEN

You need real, fresh roles, not a stale API. I pull LinkedIn through a scraping API (I use BrightData's Datasets product). The thing nobody warns you about: discovery is asynchronous.

FLOW · SCRAPE

```
-- DON'T expect the data back from one call. you don't.
-- it's three steps, and a scrape can take ~5-7 minutes.

1. TRIGGER   POST /trigger?dataset=<jobs> { keyword, location }
             returns { snapshot_id }   -- NOT the jobs

2. POLL      GET /progress/<snapshot_id>
             "running" | "ready" | "failed"

3. DOWNLOAD  GET /snapshot/<snapshot_id>?format=json
             the actual job records

-- the trick that keeps the loop fast: don't block.
-- TRIGGER in one task, then queue a "collect" task that
-- polls. if still running, it re-queues itself in 2 min.
-- the heartbeat does the waiting -- your code never sleeps.
```

> **Why it works.** The naïve version calls the scraper and waits — and either times out or jams the whole loop for seven minutes. Splitting it into *trigger* and *collect* tasks turns a long blocking call into two quick ones, and the queue's *not_before* handles the wait for free. This two-phase shape is the single most useful thing in this whole drop: any slow external job fits it.

04 The Judgment

Score every role against you — honestly.

USE IT WHEN

Once roles are flowing in. A raw feed is noise; the value is the filter. Give the model a one-paragraph profile of you (store it in a row you can edit), then have it score each role.

PROMPT · FIT SCORE

```
Score each job 0-100 for THIS candidate. Be honest
about gaps -- a low score is more useful than a kind one.

CANDIDATE:
<paste the one-paragraph dossier -- background, the
roles you actually want, your hard constraints,
location, the bar that makes something worth it>

JOBS:
<the slim list: company, title, location, summary>

Return JSON only:
{ "scored": [ {
    "company":"", "title":"", "url":"",
    "fit_score": 0-100,
    "gaps": ["what you'd be missing"],
    "why": "<one honest line>"
}]}
```

> **Why it works.** The model is a cheap, tireless screener — but only if you tell it what good means for *you*, not in general. The dossier-in-a-row is the lever: edit one paragraph and the whole board re-ranks on the next run. Forcing *gaps* and a one-line *why* keeps it from flattering you — and gives you the talking points for part 05.

BONUS · THE PART THAT CLOSES THE LOOP

+ The Reach

Draft the outreach — into a queue you approve.

USE IT WHEN

For the roles that score high. The agent finds the right people, drafts a tailored note to each — and then stops. It never sends. Every draft lands in a review table for your yes.

PATTERN · HUMAN IN THE LOOP

```
-- 1) find people (a targeted public search by company),
-- then draft in YOUR voice -- lead with a real reason.

Draft a short email (<110 words) and a shorter LinkedIn
note (<70 words) to each contact. Reference the role and
the candidate's specific fit. Calm, no hype. JSON only.

-- 2) write each draft to a "proposals" table, status=pending.
insert into proposals (action, summary, draft, status)
   values ('send_email', '<who>', '<the draft>', 'pending');

-- the send rail is deliberately NOT wired. nothing leaves
-- until you flip a row to 'approved'. the agent proposes;
-- you dispose. that gate is the whole trust model.
```

> **Why it works.** An agent that can send on its own is a liability the day it's wrong. Routing every action through a *pending → approved* queue gives you all the leverage of automation and none of the blast radius. You wake up to a stack of ready-to-go drafts, not a sent folder you have to apologize for.

THE WHOLE LOOP

Four parts, one cadence: *wake → pull → score → file → draft → sleep.*

A database that remembers. A cron that wakes it. Eyes that see the market. A judgment that ranks it against you. Wire those into one loop and you stop hunting for work — you build the thing that hunts for you, and check its findings over coffee.

— Hunter

FOLLOW THE BUILD

@huntergrant.dev

Building a system you want a second set of eyes on? That's the work. The door's open.