Episode Book

EPISODE #5870

Homebox Fork's Two Backend Scripts Need a Real Job Runner

PUBLISHED	11 October 2026 at 14:11 UTC
DURATION	24:27
LISTEN	myweirdprompts.com/episode/homebox-backend-job-orchestration-69442/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.3
SOURCES	11 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:40 UTC

SYNOPSIS

Daniel's Homebox fork added two backend scripts the upstream project was never designed to host: one converts new images to WebP and rewrites database references, the other sends images through a vision model to extract serial numbers without overwriting existing data. Both are incremental and both are deferred so the fast path stays fast. The problem is what runs them. Upstream Homebox ships a background runner with a name, an interval, and a function — no queue, no persistence, no retry, no record. Cron silently skips overlapping runs and multiplies jobs across instances. The episode walks through the standard three-part answer (cron, workers, queues), leader election via Redis or Postgres advisory locks, why CPU-bound encoding and I/O-bound vision calls need separate pools, and what actually exists in the self-hosted orchestration space: Dagu, Cronduit, Cinnamon, and a long tail that mostly ends in a web UI bolted onto crontab.

TRANSCRIPT



Corn

There's a particular kind of lie a self-hosted app tells you. It's the one where everything works, you're the only user, and nothing has ever run twice by accident. Then you add a second job, and a second instance, and the whole thing starts quietly doing the same work three times.



Herman

That's the gap. Works on my machine, versus works while the app is running.



Corn

Daniel's fork of Homebox is the machine in question. He tore it apart, changed the stack, bolted on AI features, added storage units as first-class entities, and now he's got two backend scripts that never belonged in the original project. One walks the image library, converts what's new to WebP, updates the references, deletes the original. The other sends images through a vision model to pull out serial numbers and whatever else it can read, with one rule: it can't overwrite anything already there.



Herman

Both incremental. Both deliberately deferred so the fast path stays fast. Creating an asset and reading an asset should never wait on an image transcode or a phone call to a vision model.



Corn

Which is right. The question is what actually runs them. Cron is the obvious answer and also the answer that crunches, because everything fires at once and nothing checks whether the last run finished. So what he wants to know is whether there's a real orchestration layer for this. Something that labels jobs, groups them, maybe gives you a web UI for the backlog. And underneath that, the bigger one: how do you run maintenance against data that's changing while you're working on it.



Herman

The scripts aren't the hard part. The scheduling around them is.



Corn

Start with what he's forking away from, because that explains why the scripts had no home.



Herman

Homebox gives you almost nothing here. There's a file in the backend, bgrunner.go, and it defines a struct with three fields: a name, an interval, and a function. That's it. The start method runs the function immediately, then loops on a timer, re-running it every interval until the context gets cancelled.



Corn

No queue.



Herman

No persistence, no retry, no record that it ever ran. And the work inside it is small. There's a notifier that emails people about maintenance due today, and a version check that hits GitHub for the latest release. That's the whole background surface of the upstream project.



Corn

So when Daniel wrote scripts that convert files and call a vision model, there was nowhere to put them. He wasn't ignoring an extension point. There isn't one.



Herman

And that's the honest framing of his question. He isn't looking for a Homebox feature. He's asking what the rest of the world does, because Homebox never had to answer it.



Corn

The world splits it into three things. Cron jobs, which are scheduled, run, and exit. Background workers, which stay alive and react to events. And message queues, which separate whoever produces the work from whoever does it.



Herman

And the useful part is that you almost never pick one. You pick all three and wire them together. Cron wakes something up, it drops work on a queue, a worker picks it up, and if it fails it goes back on the queue with a delay.



Corn

So the pattern's not the interesting bit. The interesting bit is what goes wrong when you skip the queue and just let cron do everything.



Herman

The failure is documented plainly. If a previous run hasn't finished when the next one is due, the next one is skipped. Not queued, not delayed. Skipped. And if the process is still running when the timer comes round again, the following runs never start at all.



Corn

It silently stops. No error, no alert. Your nightly job just isn't happening anymore.



Herman

And that's with one instance. Add a second and you get the fun version. Say you've got three instances of the API running, and each one initializes the same schedule when it boots. Now you have three copies of every job, all firing at the same moment, all fighting over the same rows in the same database.



Corn

Three nightly reports.



Herman

Three of everything. Your daily digest goes out three times. Your image conversion walks the same batch of files three times, and the first one to finish rewrites the references while the other two are still holding a path to a file that's now gone. That's not a slow job, that's a corrupted library.



Corn

What's the fix, in the standard telling?



Herman

Leader election. You take a distributed lock, either Redis with Redlock or a Postgres advisory lock, and only the instance that holds the lock runs the schedule. The other two sit and wait to take over if the leader dies.



Corn

A single lock.



Herman

A single lock, and it's doing an enormous amount of work, because everything upstream of it assumed there was one process. Add a second and the assumption is gone.



Corn

Then there's the one that bites a homelab. You could have one instance and still ruin the app's day.



Herman

CPU starvation. A job that encodes images will eat the cores the HTTP handler needs. The request thread is sitting there waiting its turn while your transcode finishes, and every user of the app feels it as latency. That's not a scaling problem, that's a Tuesday.



Corn

And the guidance there is specific. CPU-bound work runs at concurrency one per core. So on a four-core box, four image jobs at once, and no more.



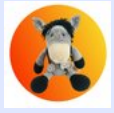
Herman

I/O-bound work is a different number entirely. Something that spends its life waiting on a network call can run ten, twenty, fifty at a time, because the CPU is idle while it waits. The cores aren't the constraint.



Corn

Which is interesting for Daniel, because his two jobs are opposite kinds of work. The WebP conversion is encoding. It's CPU.



Herman

Image encoding is as CPU-bound as it gets on a home server. And the AI enrichment is a phone call. You send bytes out, you wait, you get text back. Almost no CPU at all, and the limiting factor is how many requests the vision provider will take from you.



Corn

So they shouldn't be scheduled the same way.



Herman

They shouldn't even be in the same queue. If you run them through one worker pool with one concurrency setting, either you've crippled the enrichment, because you set it to four to protect the CPU, or you've starved the app, because you set it to thirty for the API calls and now thirty encodes are running.



Corn

Two pools.



Herman

Two pools, two concurrency numbers, two reasons to be running at all.



Corn

Right, so let's answer the actual question. Does the framework exist. Because he asked directly.



Herman

The closest thing I found is Dagu. It's a single Go binary. No external database, no broker, no Redis, no Postgres dependency of its own. You point it at existing scripts and describe the order they run in with a YAML file that declares the dependencies.



Corn

So it's a DAG runner over scripts you already have.



Herman

And it's explicitly positioned against the heavy end of this. The pitch is that a script shouldn't carry its own schedule. No cron parsing inside it, no retry loop, no check for whether the last run is still going. That belongs to whatever runs the script.



Corn

Which is exactly Daniel's complaint. His scripts currently know things they shouldn't have to know.



Herman

Right. And it comes with a web UI. You get the run history, per-step logs, retries, the whole record of what happened and when. It has overlap policies, so the default is to skip a run if the previous one is still active, which is the correct default almost every time.



Corn

That's the cron failure mode, solved by a setting.



Herman

There's also a catch-up window, so if the box was off overnight, missed intervals get run after it comes back. And per-step retry policies with timeouts and lifecycle hooks.



Corn

Hooks for what?



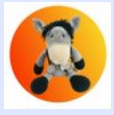
Herman

What to do on success, on failure, on exit. So the enrichment job could fire a notification or write a row when it finishes, instead of you discovering three weeks later that it died.



Corn

And the labels thing he asked about. Grouping.



Herman

That's the part that made me sit up. Worker labels let you route a step to a particular pool. You can declare that a step needs a GPU and it'll only go to workers tagged gpu=true. So the image work and the API work don't have to land on the same machine.



Corn

And they advertise media conversion as a use case.



Herman

They list ffmpeg transcoding and format conversion as a headline example. Which is Daniel's WebP job, minus the reference rewriting.



Corn

Slightly under his problem, then. He's not converting files in a folder, he's converting files that a database points at, and then rewriting the pointers.



Herman

But there's a feature for that too, and it's the one I'd have designed for him. Build workflows. A step declares its inputs and outputs by file path, and if the input hasn't changed since the last run, Dagu reuses the output and skips the work.



Corn

So it remembers what it's already done.



Herman

It remembers by content, not by a timestamp. That's a real difference. A file gets touched but not changed, and a timestamp-based job redoes the work. This one doesn't.



Corn

Which is the increment-over-what-changed behaviour he's hand-rolling right now.



Herman

Hand-rolling and getting right, which is worth saying. His design has the increment logic and the safety rule already. The tool would replace the plumbing, not the thinking.



Corn

What else is out there, and what's wrong with it.



Herman

Cronduit is the other one that'll get recommended. Rust, MIT licensed, one point two point one as of May, Docker-native with a web UI and job tags you can filter on the dashboard.



Corn

Tags with filter chips is a nice touch. That's his labelling requirement, done.



Herman

Except it mounts the Docker socket. Which is root-equivalent on the host, so anything that gets into the web UI effectively owns the machine.



Corn

And the web UI ships unauthenticated.



Herman

In version one, yes. They say so themselves. It's built as a single-operator homelab tool and the security model reflects that. Which is a interesting trade-off, because the convenience of a browser dashboard is the entire reason you'd install it, and the attack surface arrives in the same box.



Corn

A dashboard someone else can reach is a shell someone else can reach.



Herman

That's the cautionary tale in the whole landscape. The moment you want a web UI for backend jobs, you've built an interface to code execution, and the interface is the interesting target.



Corn

What's the third one.



Herman

Cinnamon. TypeScript and Bun, runs on BullMQ with Postgres and a Hono API behind a React dashboard. Jobs are declared in a config file, and you can trigger them by CLI, by API, or by cron. It's multi-tenant, which none of the others really are.



Corn

And beyond those?



Herman

A long tail. Kestra is the heavy event-driven one, language-agnostic, aimed at enterprise volume. There's Dagychu, Chronoverse, Tikeo out of Rust with RBAC and OpenTelemetry, a single-binary one called orchestrator that keeps its state in SQLite, and cronmanager, which is a web UI bolted onto the Linux crontab you already have.



Corn

Which is the smallest possible version of the idea.



Herman

It is, and I'd bet it's what most people actually end up running. A UI over crontab is a real improvement and it costs you almost nothing in operational surface.



Corn

So does the thing he asked for exist. Direct answer.



Herman

No. Not the thing as described. There is no off-the-shelf tool that's purpose-built for async backend enrichments living inside one app, with a web UI attached to that app's data. What exists is a category of general-purpose self-hosted orchestrators, and they run alongside your app rather than inside it.



Corn

Every one of them is a second system.



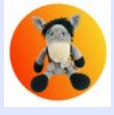
Herman

Every one. And I looked at this from a couple of angles. Web search, and the developer forums, and nothing surfaced that's solving this as an embedded feature. Which is either a hole in the market or a sign the pattern is wrong.



Corn

Hold that thought, because it turns out to be the whole episode.



Herman

It does. Because the question shifts. Not what tool do I run, but does the work belong in the process at all.



Corn

Which is the deeper half of what Daniel asked. Overnight jobs against data that can change while they run.



Herman

And the pattern underneath all of it is idempotency. If a job can only safely run once, you have a problem, because almost every queue is at-least-once. Your job will run twice. Not might. Will.



Corn

Why is that the design.



Herman

Because the alternative is worse. To guarantee exactly once you have to know for certain that a job didn't run, and the only way to know that is to have missed an acknowledgment, and the moment an acknowledgment gets lost you can't tell the difference between the job never happening and the confirmation never arriving. So the system does the safe thing and runs it again.



Corn

So the job has to survive being run twice.



Herman

Which means either you keep a record of what you've processed and check before you act, or you design the operation so running it twice is the same as running it once.



Corn

And Daniel's already done the second one, whether he'd call it that or not. His enrichment rule is that it can't override existing data. It only writes into fields that are empty.



Herman

Which means if it runs again, the second pass finds the field already filled and does nothing. It's naturally idempotent. That's not a nice property of his design, it's the property that makes the design safe to put behind a queue at all.



Corn

What about the WebP job. That one deletes things.



Herman

Convert, update the reference, delete the original. Run that twice and the second run looks for an original that's gone and finds the WebP already in place. So it does nothing the second time, as long as the check is on the file existing rather than on a flag saying the job ran.



Corn

If the check is a flag, you've got a window. Job starts, writes the flag, crashes before it deletes, and now nothing ever cleans it up.



Herman

And that's the difference between a job that's idempotent by design and a job that's idempotent by luck.



Corn

What else does the discipline include.



Herman

Backoff with jitter. When a job fails you retry it, but you don't retry it at a fixed interval, and you don't retry it at a pure exponentially increasing interval either, because if a hundred jobs all fail at the same moment, they all retry at the same moment, and you've built a stampede.



Corn

So you randomise.



Herman

You randomise the delay slightly. Pure backoff synchronises. Jitter desynchronises. It's a small change that stops a bad minute from becoming a bad hour.



Corn

And where do jobs go when they've failed enough times.



Herman

A dead-letter queue. After the retries are exhausted, the job lands somewhere it can be looked at instead of evaporating. Which is the difference between the job failed and the job failed and nobody noticed for three weeks.



Corn

That second one is how you lose a month of image conversions.



Herman

It's how you lose a month of anything. A silent failure is indistinguishable from a job that has nothing to do.



Corn

Priorities.



Herman

Three named levels. Critical, normal, low. Not a numeric range you invent, because nobody can remember whether seven is more important than three a year after the code's written.



Corn

Daniel's fast path is critical and it's already critical, because it's synchronous. He's asking the right question by keeping it out of the queue.



Herman

Exactly right. Reads and asset creation never go in the queue. WebP conversion is low, and enrichment is low, and if there's ever a user-facing action that needs enrichment immediately, that's a different job with a different priority.



Corn

Then the operational layer, which is the part nobody builds until something breaks.



Herman

Watch the queue depth. If it's above a thousand for two minutes, add capacity. If it's under a hundred for ten, take capacity away. Those are the published numbers from the queue vendors and they're sensible defaults rather than laws.



Corn

And alerts.



Herman

Queue depth above five thousand for five minutes. Dead-letter queue above zero, which is the one that should page somebody, because a non-empty dead-letter queue means something is failing and you haven't dealt with it. And overall failure rate above five percent.



Corn

A dead-letter queue above zero as a page is aggressive.



Herman

It is. And it's the right call for a homelab, because if you look at your dead-letter queue once a month, you've built a queue whose entire job is to hold things you'll never read.



Corn

So back to the live-data problem, which is the part he framed best. Maintenance running while the app is operational and the data can change under it.



Herman

The first rule of it is the one we already landed on. Separate the worker from the API server. If the job shares a process with the request handler, a CPU-heavy job is a latency incident for every user.



Corn

And that's not hypothetical for him. The transcoding job will do exactly that.



Herman

The second rule is that the job has to tolerate the world moving. You select a batch of images to convert, and while you're halfway through, someone uploads a new one and someone else edits an asset that one of your images belongs to. Your batch has to not care.



Corn

So you work off a snapshot of what needs doing, and you re-check before each write.



Herman

Re-check the row before you touch it. It's the same optimistic concurrency you'd use anywhere else. If it moved, skip it, it'll be in the next batch.



Corn

The temptation is to lock everything for the duration.



Herman

And that's how a maintenance job becomes an outage. You lock the images table for twenty minutes while you transcode, and the app is read-only for twenty minutes, and you've built a scheduled downtime and called it a background task.



Corn

Then there's a fork in the road that his architecture is standing right on.



Herman

His instinct is to keep the jobs in-process, and the instinct is correct for exactly one instance. That's the homelab reality. One box, one process, one user, and in-process is simpler and has no second system to operate.



Corn

Until it isn't.



Herman

The moment you run two instances, in-process breaks. Breaks. Every instance initializes the same schedule and you get duplicate work racing for the same rows.



Corn

So either you add leader election and stay in-process, or you pull the jobs out into their own process and there's nothing left to duplicate.



Herman

And leader election is a real answer. A Postgres advisory lock is six lines of code and it solves the duplication. It doesn't solve the CPU starvation, and it doesn't give you a UI, and it doesn't give you run history, but for one instance and a low-stakes job, it's honestly fine.



Corn

Then say the uncomfortable part.



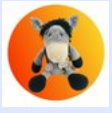
Herman

The uncomfortable part is that adopting Dagu means running a second system next to your app. Which is the exact thing Dagu's own marketing criticises about the heavy orchestrators. The line is that you wanted to schedule some jobs and now you're operating a second system, and the orchestrator lives inside the code it was supposed to serve.



Corn

So it's a criticism it also earns.



Herman

Every external scheduler earns it. You either embed orchestration and accept the scaling ceiling, or you externalise it and accept the operational overhead. There's no version where you get the UI and the run history and the routing labels and also nothing new to run.



Corn

Which is the answer to his question, really. Here's the trade, pick your side.



Herman

And there's one more thing I couldn't answer. The literature on backfilling against live tables, the online schema change tools, gh-ost and the MySQL ones, that whole thread is about the same problem he's describing. Rewriting data while the app is using it. I didn't get to it.



Corn

So flag it and move on.



Herman

Flagging it. If you're doing what Daniel's doing, backfilling a column on a live table, that's a whole literature and it's worth the read.



Corn

I want to go back to something you said, because it's the part I don't think he'll like.



Herman

Go on.



Corn

Every guide says separate the worker from the API server. His whole design keeps them together, and he's done it deliberately, because together is simpler. The guides aren't wrong. Neither is he. They're answering different questions.



Herman

One instance is a deployment decision, not a philosophy.



Corn

At one instance, in-process with a lock is correct. At three, it's a bug report.



Hilbert

Herman, when you said the transcoding job will starve the request handler, that's not a software problem. The machine has a limit and you're pretending it doesn't.



Herman

I'm not pretending it doesn't, I'm saying you can schedule around it.



Hilbert

My uncle ran a printing operation out of a shed behind his house. Had a clipboard. Every job that couldn't run during business hours got written on it, and the order had nothing to do with when someone asked for it. It was how much ink the job used. High-ink jobs went at the bottom of the list, because the press had to cool down in between. He called it the cooling-off list.



Corn

Your uncle had a queue.



Hilbert

He had a clipboard. The press wasn't really a press. It was a washing machine he'd modified, and the cooling-off was because the motor would overheat and start smelling like a barbecue. Which is why the ink mattered.



Corn

The ink determined the motor temperature.



Hilbert

The ink determined how long it stayed hot. That's the whole list. Two columns on the clipboard. Jobs that can wait forever, and jobs that will ruin the press if you run them cold.



Herman

Where does the enrichment go? The vision model.



Hilbert

The AI doesn't care. It's not a machine, it's a phone call. It's the paper that's the problem.



Herman

So the conversion goes in the second column.



Hilbert

Conversion goes in the second column and it goes last, because if you run it during the cooling-off period the press makes a sound like a goose being stepped on and then it never works right again.



Corn

He ran a job during the cooling-off period.



Hilbert

Once. Nineteen seconds into it.



Herman

That killed the press.



Hilbert

It made the goose sound. After that it ran but the registration was off by about a millimeter, and he never fixed it, and every job he did afterward was off by a millimeter. He still took the work. People couldn't tell. He could tell.



Corn

A millimeter.



Hilbert

Everything. Until he stopped.



Herman

I don't know what to do with any of that.



Hilbert

I've got one on the enrichment job too, if you want to go deeper.



Corn

No, I think that's the right depth.



Corn

The misconception worth naming is the one I had walking in. That cron is the boring, safe option and queues are the complicated one.



Herman

Cron is the one that fails silently. It skips, it doesn't error, and it duplicates the moment you add a second instance. The queue is the boring one. It fails loudly and it tells you.



Corn

What we found, for you, Daniel, is that nobody sells your exact thing. General orchestrators exist, Dagu's the closest, and it runs next to your app rather than inside it.



Herman

The open question is whether that's a gap worth filling. An embedded job runner with a UI and a queue is a real product if enough self-hosted apps end up needing one. Or it's a sign that anything that needs those features has already outgrown being embedded.



Corn

The answer changes with the deployment. One instance, keep it simple, take the lock and move on. Three instances, you don't have a choice.



Herman

The scripts were never the hard part.



Corn

That's been My Weird Prompts. Hilbert Flumingtop produces the show.



Herman

If you're running something self-hosted with a pile of deferred jobs, send us your own prompt on Telegram at t dot me slash MWP listener bot.



Corn

We'll be back soon.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **original Homebox repo (structure, README, Dockerfile) [primary]**
   https://github.com/hay-kot/homebox

2. **Homebox background task runner source [primary]**
   https://github.com/sysadminsmedia/homebox/blob/main/backend/app/api/bgrunner.go

3. **Homebox background service source [primary]**
   https://raw.githubusercontent.com/sysadminsmedia/homebox/main/backend/internal/core/services/service_background.go

4. **Homebox backend docs, last updated 2026-10-04 [primary]**
   https://homebox.software/en/contribute/development/backend/

5. **Dagu docs, last updated 2026-10-08 [primary]**
   https://docs.dagu.sh/

6. **Dagu repo (4,306 stars, GPL-3.0) [primary]**
   https://github.com/dagucloud/dagu

7. **Cronduit repo (v1.2.1, 2026-05-20) [primary]**
   https://github.com/SimplicityGuy/cronduit

8. **Cinnamon repo (v0.3.1, 2026-04-04) [primary]**
   https://github.com/ianuriegas/cinnamon

9. **Homebox homepage (50MB idle) [primary]**
   https://homebox.software/en/

10. **Railway cron/workers/queues guide, dated 2026-03-30**
    https://github.com/railwayapp/docs/blob/76db91a4/content/guides/cron-workers-queues.md

11. **Kanopy Labs background jobs guide, 2026-07-06**
https://kanopylabs.com/blog/background-jobs-architecture-queues-cron-workers