Episode Book

EPISODE #6099

Code-First Workflow Runners: Prefect, Temporal, Dagster

PUBLISHED	18 October 2026 at 18:16 UTC
DURATION	23:19
LISTEN	myweirdprompts.com/episode/code-first-workflow-orchestrators-07963/
HOSTS	Herman, Corn, Larry, Hilbert
SCRIPT MODEL	anthropic/claude-opus-5.5
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.10
BOOK GENERATED	19 October 2026 at 17:43 UTC

SYNOPSIS

If you've outgrown n8n's canvas and want your workflows versioned in git with real observability, this episode maps the code-first workflow orchestrator field. Prefect is the closest match to a long-lived Python script runner — flows, tasks, workers, and a server you can self-host. Temporal offers durable execution where a workflow can sleep for thirty days and resume on the exact line. And Dagster got a lot more interesting in July 2026 when Prefect announced it was acquiring Dagster Labs. We compare all three, explain what "robust" actually means as a checklist, and walk through what a five-node n8n canvas becomes as a sixty-line Python file.

TRANSCRIPT



Herman

I have wanted this one for months. Actual months.



Corn

He has a folder. I've seen the folder.



Corn

Here's where Daniel's at. The podcast itself gets generated on Modal, and for a while now he's been hunting for the self-hosted version of something hard to even name. A long-lived Python script runner. Scripts, containers, jobs that run for an hour, with logging, observability, controlled environments, and enough of a GUI to see what's going on. He's got N eight N, which he likes, and which has limits he's hit. His guess is that Prefect is the closest match, and he wants the robust end of the field, ranked, with the pros and cons.



Herman

He's right about Prefect, for what it's worth. But I think the bigger thing he's asking for is a name for the category, because once you can name it you can shop for it.



Corn

Go on then. Name it.



Herman

The name the industry uses is workflow orchestrator, which is terrible, because N eight N calls itself that too. What Daniel means is narrower. You've written the logic in code. That code lives in git. A server somewhere keeps the state and the history of every run. Workers that you control pick up the jobs and run them. And the web interface is a window, not an editor. You watch the work there, you retry it there, you never build it there.



Corn

So the GUI got demoted.



Herman

Demoted to dashboard, yes. In N eight N the canvas is the program. In this category the program is a Python file, and the canvas just draws you a picture of what it did last night.



Corn

Which is the whole reason you'd graduate. Not because N eight N is bad, but because a workflow you can't diff is a workflow you can't trust at two in the morning.



Herman

And that's the line I'd draw. If you want version control, code review, tests, an AI agent writing the thing for you, you've already left the canvas tools. The question is just which one of the code-first runners you land on.



Corn

Before you hand me a list, Herman, I want to guess the order. Prefect first, because Daniel said it. Then something enormous that you'll love and nobody needs.



Herman

That's... unkind and roughly accurate. Prefect first, then Temporal, which is the enormous thing, and then Dagster, which got a lot more interesting in July.



Corn

What happened in July?



Herman

Prefect announced it was acquiring Dagster Labs. July thirteenth. The two biggest of the post-Airflow orchestrators under one roof. Dagster's founder wrote that Dagster stays its own product and keeps its open source licence, so it isn't vanishing. But it changes the shape of the field.



Corn

Two rivals merging, and both promising nothing will change. I've heard that wedding toast before.



Herman

Prefect first, though, because it's the closest to what Daniel described. You write ordinary Python. You put a flow decorator on the main function and a task decorator on the pieces, and that's most of the learning curve. Each task gets retries, caching, timeouts, logging, without you writing any of it.



Corn

And the self-hosted part?



Herman

Three pieces. The Prefect server, which is the API plus the web interface. A Postgres database, where all the state lives. And a worker, which polls something called a work pool, claims whatever's scheduled, and runs it. Docker container, subprocess, Kubernetes job, whatever you configured. Apache two licence, all of it.



Corn

The worker's the bit I'd forget.



Herman

Everyone forgets it. A Prefect server with no worker is a beautiful control room over an empty factory. Your schedules sit there, pending, forever, looking very organised.



Corn

That's most of my to-do list.



Herman

The thing I like for Daniel specifically is that Prefect doesn't care how long a run takes. A ninety-minute job is just a flow that takes ninety minutes. You see it in the UI as a run with its state, its logs, its task timeline, and if step four of seven blew up you retry from there.



Corn

Here's my doubt, though. He uses Modal. Modal hands you the computer as well. Self-hosted Prefect doesn't give you a GPU. It gives you a scheduler with good taste.



Herman

Hm. That's fair, and it's the thing people blur constantly. Modal is compute. Prefect is orchestration. They're different layers. Prefect can actually push jobs onto Modal, there's a Modal push work pool, but... I'd better be precise. Push pools are a Prefect Cloud feature. Self-host the server and you're back to running your own workers on your own boxes.



Corn

So the self-hosted Modal doesn't exist. It's self-hosted Prefect plus a machine you own.



Herman

Plus a machine you own, yes. Which, for long-running Python, is usually what you wanted anyway.



Corn

Let me make the step up concrete, because Daniel's really asking whether he can leave N eight N without starting over. Take a typical workflow. A form gets submitted, the webhook fires, a model summarises the text, a row goes into a Google Sheet, a message goes to Telegram. Five nodes on a canvas. What does that become?



Herman

One Python file, maybe sixty lines. A flow called something like handle submission, and four tasks inside it. Summarise, write the row, send the message, and a little validation step N eight N never made you write. Each task gets its own retries, so if Telegram is down for a minute, the sheet row isn't written twice when it tries again.



Corn

Is there a converter? Export the workflow, get Python out the other end?



Herman

Not an official one that I know of. But the export is a JSON file describing every node and its settings, and handing that to a coding agent with the instruction rewrite this as a Prefect flow works surprisingly well. The logic was never the hard part. The plumbing around it is.



Corn

The plumbing being the four hundred little connectors.



Herman

That's where people get it backwards. You mostly don't need an equivalent connector. Every service those nodes talk to has an official Python library, Google, Slack, Telegram, the model providers. The node was a wrapper around that library with a nice form on top. Prefect does publish integration packages for the big clouds and a few others, but for most steps you just import the vendor's own client.



Corn

So the connectors were the training wheels, and the bicycle was there all along.



Herman

What you really lose is the authentication handling. N eight N does the Google sign-in dance and keeps refreshing the token for you. In code you set that up yourself once, and you store the keys as secrets in Prefect rather than in the script.



Corn

And the trigger? The webhook was the whole front door.



Herman

That's the other gap. A schedule is trivial, every five minutes, every night at two. An incoming webhook on a self-hosted Prefect server is less polished, as far as I know the hosted webhook feature lives in the paid cloud. The common pattern is a tiny web endpoint that receives the request and tells Prefect to start a run of the deployment. Or, and I like this one, you keep N eight N as the front door.



Corn

Keep the thing you're leaving.



Herman

Keep it for what it's good at. It catches the webhook, does the trivial reshuffling, then makes one HTTP call that starts the Prefect flow. Anything long, heavy or important moves into code. Would you migrate everything in one go, or bit by bit?



Corn

Bit by bit, worst offender first. Whichever workflow you've opened the execution log for most often this month. That's the one that's earned its own Python file.

L

Larry

Are your workflows suffering from orchestration fatigue? Do your Python scripts wake up confused, unsure of who scheduled them or why? Introducing Decorator Supreme, the first dietary supplement for code! Just sprinkle a teaspoon of finely ground decorators over any function, and watch it retry itself, cache itself, and file its own tax return! Customers report flows so durable their grandchildren still run them! Decorator Supreme. Because your tasks deserve a work pool with a diving board! Not evaluated by any compiler. May cause infinite retries, spontaneous Kubernetes, and mild YAML. Do not apply to production. Results not typical, or possible.



Herman

Decorators aren't actually something you can grind...



Corn

Herman. No. Where were you?



Herman

The rough edges on Prefect, because there are a few, and Daniel asked for cons. The open source server has thin access control. There's a basic authentication setting where the whole UI asks for one username and password string, and that's about it. Proper roles, teams, API keys, those live in the paid cloud.



Corn

One password for everything. Very household.



Herman

For a one-person setup behind a tunnel, honestly fine. Second con: Prefect's history is a bit of a shapeshifter. The jump from version one to version two was a near rewrite, and three changed things again. People who adopted early remember that.



Corn

And the third?



Herman

It's still another service to run. Server, database, worker. Less than Airflow, more than a cron job. That's the price of the dashboard.



Corn

Daniel said controlled environments, though, and I think that's doing more work in his question than the dashboard is. What does Prefect actually give you there?



Herman

More than people expect. Every deployment, which is Prefect's word for a flow plus its schedule plus where it runs, gets defined in a file that sits in the repo next to the code. You can point each one at a Docker work pool and give it its own image. So the transcription job runs in an image with the heavy audio libraries, the backup job runs in a tiny one, and neither can break the other by upgrading a package.



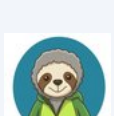
Corn

That's the bit that bit him with scripts in folders, I'd bet. One shared virtual environment, ten jobs, and the eleventh install quietly ruins three of them.



Herman

Every homelab has that story. And the deployment file being in git is the version control half of his ask. You change a schedule in a pull request, not in a web form at midnight. Roll it back the same way.



Corn

Here's what I'd want to know before I trusted any of these, Herman. What does robust actually mean? Because every vendor page uses the word and none of them mean the same thing.



Herman

Hm. Let me try a checklist, and you tell me if I'm cheating. One, the state of a run lives outside the process running it, so a crash doesn't erase the evidence. Two, retries happen per step, not per whole script, so a flaky upload at minute fifty doesn't redo the first forty-nine. Three, there's a history you can query later. Four, you can cap how many of a thing run at once. Five, secrets and environments are managed, not pasted.



Corn

That's five things and you said a checklist, which in your mouth means eleven.



Herman

I cut it to five for you. I had nine. Anything you'd add?



Corn

Something that tells a person when it fails. A run history nobody reads is a diary.



Herman

Fair. Every one of these can fire an alert on failure, but you have to switch it on, and almost nobody does until the first silent week.



Corn

N eight N passes about two of those.



Herman

Two and a half, being generous. Prefect passes all five comfortably. Temporal passes all five and then invents a sixth.



Corn

Fine. Temporal. The enormous thing. Convince me Daniel needs it.



Herman

I'll convince you he should understand it, which isn't the same. Temporal is the most robust thing in this whole space, and the reason is a single idea. Durable execution. Every step your workflow takes is written to an event history on the server. If a worker dies halfway through, another worker picks it up, replays the history, and carries on from the exact line it was on.



Corn

The exact line. Not the start of the task.



Herman

The exact line. You can write a workflow that sleeps for thirty days, and the server keeps the promise while no process is alive at all. Banks, ride-hailing companies, payment systems run on this for that reason.



Corn

There's a catch. There's always a catch with your favourite things.



Herman

There's a catch. Your workflow code has to be deterministic. No reading the clock directly, no random numbers, no network calls in the workflow itself. Anything with side effects goes in what Temporal calls an activity, and the workflow just orchestrates the activities.



Corn

So I can't just paste in my script.



Herman

You can't just paste in your script. You restructure it. That's the tax.



Corn

What do you get on the GUI side? Because that was half of Daniel's ask.



Herman

A good web UI, actually. Every workflow, its status, and the full event history, step by step, with inputs and outputs. For debugging something that went wrong at three in the morning, it's the best window in the category. MIT licence, self-hostable, the server runs in Docker, the Python SDK is solid.



Corn

Herman, I'm going to predict you, since you predicted nothing. You are about to say Daniel should use it.



Herman

I... wasn't going to. What would you do with it, though, honestly? You've watched him build the podcast pipeline.



Corn

I'd keep it for the one job where losing state costs real money. Publishing an episode, charging a card, anything where running twice is worse than not running at all. For a pile of nightly scripts it's a forklift for moving a sofa cushion.



Herman

That's where I land too, mostly. I'd add that Temporal has no opinion about schedules being pretty or scripts being easy to drop in. Prefect gives you a dashboard of your jobs. Temporal gives you a court transcript of your jobs.



Corn

There's another con you've skipped, and I think you skipped it because it's boring.



Herman

Go on.



Corn

Versioning. If a workflow can sleep for thirty days, and I change the code on day twelve, what wakes up on day thirty?



Herman

Ah. Yes. That's the sharpest edge Temporal has, and it's the price of the replay trick. When a worker replays the history, the code has to make the same decisions it made the first time. Change the logic underneath a running workflow and the replay no longer matches. Temporal has versioning tools for this, you mark where the old path and the new path split, but you have to think about it every time you change a long-running workflow.



Corn

So deploying a fix becomes a small act of archaeology.



Herman

For long-lived workflows, a bit, yes. For jobs that finish in an hour it barely matters, because nothing old is still asleep when you deploy. Which is another way of saying Daniel's use case doesn't need the thing that makes Temporal hard.



Corn

I'll take that as a concession.



Herman

Take it as a fact. I still love it.



Corn

Dagster, then. Where does it fit, now it's half Prefect?



Herman

Dagster thinks in a different shape, and that's both its strength and why it's third for Daniel. Its central idea is the asset. You don't describe tasks, you describe the things that should exist, a cleaned table, a trained model, a transcript, and how each is derived from the others. Dagster works out what to run to make them fresh.



Corn

So it's a build system for data.



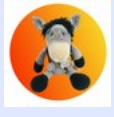
Herman

Pretty much, yes. And the UI is gorgeous for that. You get a lineage graph of every asset, when it was last materialised, what it depends on. Asset lineage ships in the open source version. Roles and single sign-on are the paid Dagster Plus.



Corn

Here's my problem with it for this. Daniel's jobs are verbs. Run the backup, render the thing, push the file. Dagster wants nouns.



Herman

That's the right objection. Though Dagster does one thing on environments better than the others, and I don't want to skip it. Your code runs in what it calls code locations, separate processes or containers from the Dagster server itself. A broken import in one project can't take down the UI or the other projects. Each one has its own dependencies.



Corn

Isolation as a default rather than something you configure. I like that more than I want to.



Herman

You can make Dagster run verbs. People do. It just feels like wearing someone else's coat. If his pipeline were really about keeping a set of datasets current, the episode catalogue, the transcripts, the embeddings, Dagster would be the most elegant of the three.



Herman

Would you bet on it right now, though? After the acquisition?



Corn

Not with a homelab. Two good teams with money behind them, probably fine. Probably is doing a lot of lifting in that sentence.



Herman

That's about where I am. Give it a year and see which product the roadmap is actually being written for.



Corn

For completeness, because you'll sulk otherwise. Airflow. Windmill.



Herman

Airflow is the grandparent. Huge ecosystem, every connector imaginable, still the safe hire at a big company. For one person it's a lot of machinery. Windmill... it's there. It runs scripts in lots of languages, it builds a little form for each script from its parameters, and it's fast. But it leans toward building internal tools, and Daniel's never warmed to it.



Corn

So it's not a pick.



Herman

It's not a pick. If you want the most robust code-first runner for long-running Python with a dashboard, it's Prefect, Temporal or Dagster, and they rank differently depending on what you're protecting.



Corn

Mine ranks by what breaks least at a weekend.



Hilbert

Kessler-Brandt mechanical segment timer. Model ST forty-eight. Thirty-nine shekels.



Corn

That's a plug-in wall timer.



Hilbert

It is a scheduler, Corn. Ninety-six pins, one for every fifteen minutes. You push a pin in, the power comes on. You pull it out, the power goes off. No database. No worker. No event history. I am moving everything onto it this week.



Herman

Moving everything how, exactly?



Hilbert

The timer sits on the windowsill, plugged into the extension behind the fridge. Into the timer goes an old laptop. The laptop is set to switch itself on the moment it gets power. It boots, it runs every script in one folder, top to bottom, and when the pin runs out, it dies. That is the retry policy. Tomorrow it tries again whether it likes it or not.



Hilbert

For logging I have bought a receipt printer. Thermal. Sixty shekels second-hand from a falafel place on Agrippas that went over to tablets. Every script prints one line when it starts and one when it finishes. The roll comes off the windowsill and goes down behind the radiator. That is my observability. If I want history I follow the paper.



Corn

Following a paper trail into a radiator is still more observability than most homelabs have.



Hilbert

The second laptop is in the oven.



Herman

In the oven.



Hilbert

Ventilation.



Hilbert

The second laptop watches the first one with its webcam and records it, so I can see what the screen said when it died. Herman, you said work pool about eleven times this afternoon. This is a work pool. It is a pool of two laptops, and one of them is in the oven.



Hilbert

I had a scheduler once. Proper software. It renewed my parking permit with the municipality, one payment a year. I typed the schedule in on a Friday. I typed it every minute instead of every year. By Monday morning I held four hundred and twelve valid parking permits for one Subaru.



Herman

The thing your scheduler lacked wasn't intelligence. A concurrency limit would have stopped it at one. Prefect and Temporal both let you say only one of these may run at a time.



Hilbert

The municipality refunded eleven of them.



Hilbert

The receipt printer is honestly the nicest thing I own. It cuts the paper by itself. A little blade. Every time.



Corn

The pin can't fire every minute. That's the one real feature of a thirty-nine shekel timer. It physically can't do what your software did.



Hilbert

The timer has ninety-six pins. So far I have pushed in four. The other four hundred and one permits are in a shoebox under the printer.



Herman

Four hundred and twelve permits, and the fix is a device that cannot fire more than once every fifteen minutes. There's something to that, Corn.



Corn

There is. A run history that tells you something happened four hundred times before Monday is the feature. Not the pretty graph. The count.



Herman

And that's where the three split. Prefect shows you the count and the logs and lets you cap it. Temporal shows you every step of every one of the four hundred. Dagster asks you which asset the permits were supposed to be.



Corn

The misconception I'd want to clear up, Herman, is the one Daniel half-had going in. People think Prefect is a self-hosted Modal.



Herman

It isn't. Modal sells you the computer. Prefect decides when things run and keeps the record of what happened. Self-host it and the computer is whatever box you put the worker on.



Corn

Where I'd land for him. Prefect as the backbone, because it takes his Python close to as-is and gives him the dashboard he asked for. Temporal held in reserve for the one job where running twice is worse than not running at all. And whatever he picks, the first thing to set is a limit on how many of anything can run at once.



Herman

Learned at the cost of four hundred and twelve parking permits.



Corn

Thanks as always to our producer, Hilbert Flumingtop, for keeping the show running on something better than a wall timer.



Herman

This has been My Weird Prompts. If you want the back catalogue, it's all at my weird prompts dot com.



Corn

See you tomorrow.