Episode Book

EPISODE #6512

Build Servers Never Actually Build Anything

PUBLISHED	9 November 2026 at 20:21 UTC
DURATION	22:15
LISTEN	myweirdprompts.com/episode/self-hosted-ci-build-runners-75150/
HOSTS	Corn, Herman, Larry, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	25 cited by the research (listed at the end)
BOOK GENERATED	10 November 2026 at 10:05 UTC

SYNOPSIS

Continuous integration terminology is full of misnomers, starting with the fact that build servers rarely compile code. Evaluating Woodpecker CI, Forgejo Runner, and self-hosted GitHub Actions runners reveals how build jobs actually move from git push to execution. Job routing relies on lease-based polling queues, label matching, and outbound-only network connections, while isolation models vary wildly from hardened Kubernetes environments to host execution that can permanently destroy a system.

TRANSCRIPT



Corn

A machine that takes your code, gives you back a binary, and asks for nothing else. That's the whole pitch, and honestly it's suspiciously close to my ideal of computing.



Herman

Minimal interaction, maximal output.



Corn

Daniel's with me on this one, I think. He sent us a prompt about the build half of CI — separated from deployment entirely. You push code to a repo, a box you own picks up the job, compiles it, and, his words, hands you back a link to the finished package. No deploy step, no vendor cloud. But he's stuck on something upstream of the tooling: what is that thing even called? Build runner? Build farm? Build server? CI agent? And then, once the naming is settled, which self-hostable tools actually do just build-and-artifact without dragging a whole deployment platform along — he's thinking Woodpecker, the Forgejo runner, GitHub's self-hosted runners — how a job travels from push to runner under the hood, how the build is isolated, where the artifacts land, and how you'd pick one for a small private box.



Herman

And I love that he started with the vocabulary question, because it's not a warm-up. It's the most clarifying question in the whole subject. The words people use for this are actively misleading.



Corn

Because the obvious one lies.



Herman

"Build server" sounds like the machine that builds. It almost never is.



Herman

Take Woodpecker, which is probably the cleanest example. The docs describe two essential components — a server and an agent, plus an optional autoscaler. The server is the control plane. It receives the webhook from your forge, serves the web UI and the API, parses the pipeline YAML, and runs the queue. It never compiles anything. The agent is the process that actually executes the workflow through some backend — Docker, Kubernetes, or local — and it connects to the server over gRPC.



Corn

So if you go shopping for "a build server," you're evaluating the piece that doesn't build.



Herman

That's the trap. You end up comparing control planes when what you want is a runner plus something to hand it jobs.



Corn

Meanwhile "build farm" is worse. It sounds architectural.



Herman

It's a scale word. The old Stack Exchange answer on this defines a build server as a server running a CI application, Jenkins being the classic, and a build farm as — I'm quoting from memory — a bunch of build servers. That's it. It tells you how many, not how they're arranged.



Corn

The Jenkins era gave us a farming term for a counting problem.



Herman

The modern word is "runner" or "agent," and it's about the process, not the box. GitHub says runner, Forgejo says runner, Woodpecker says agent, CircleCI's glossary says build agent. All the same idea: the program that shows up, executes the steps, and reports



Herman (continued)

results.



Corn

And "build runner" — Daniel's candidate — is nobody's official term.



Herman

Nobody's. It's a perfectly good colloquialism and it appears in none of the primary docs. Which is fine, everyone will know what you mean. But if you're searching documentation with it, you'll get worse results than with "agent" or "runner."



Corn

So the answer to his first question is almost a translation table. The box is a build server, or honestly a build box. The process is a runner or agent. A pile of them is a farm. And the "server" in Woodpecker and Forgejo is a dispatcher wearing a server's name.



Herman

Settled. Now, if the server isn't the build machine, what are the actual options for the machine that is — and how does a job find its way to one?



Herman

The three that matter for what Daniel wants are Woodpecker CI, Forgejo Runner, and GitHub's self-hosted runner. Woodpecker first, because it's the closest fit to the picture in his prompt. It's a community fork of Drone, written in Go, Apache-2.0, actively maintained — a release at the end of October. Two-part architecture, as we said. The footprint is absurd in the good direction: about a hundred megabytes of RAM for the server at idle, thirty for an agent, SQLite by default, and the docs cheerfully note it runs on a Raspberry Pi.



Corn

Which is less machine than the fridge has.



Herman

It's what Codeberg runs for their whole forge, at scale, which is the strongest existence proof a small deployment could ask for.



Corn

One caveat on versions, since you mentioned the October release. Three point twenty point zero shipped an unintended breaking change, and the project's own release notes say skip it, use three point twenty point one. When the fix for your version is "install the next one," the next one is the version.



Herman

Forgejo Runner is a different animal. It's a fork of the old Gitea act runner, maintained by the Forgejo community, and it's for Forgejo only — GPL-3.0, current line is thirteen-something. One thing that bites people: if you go looking for it on GitHub, the forgejo runner repository there returns not found. The canonical source lives at code dot forgejo dot org. Half the wrappers and charts you'll find on GitHub are third-party.



Corn

The GitHub-shaped site doesn't have the GitHub-alternative's runner.



Herman

There's a certain poetry to it. Third, the GitHub Actions self-hosted runner itself — actions slash runner. C#, MIT-licensed, latest release late October. Runs a job from a GitHub Actions workflow on your own hardware. The catch is in the README, in the politest possible terms: they're not taking contributions right now. Security fixes, yes. Patches from you, no.



Corn

Open source the way a museum is open. You can look, you can visit, don't move anything.



Herman

Now the interesting part — how a job actually gets from a push to a runner, because the three systems do it three different ways. Woodpecker: the forge sends a webhook to the server, the server parses the YAML and enqueues tasks, and agents poll that queue over gRPC. And the queue is more sophisticated than it sounds. Poll takes an agent ID and a filter function that returns a matching score, and the whole thing runs on lease semantics — if a running task blows past its lease, it expires and gets resubmitted. If the task ended up assigned to a different agent, that's a distinct error.



Corn

Meaning an agent can die mid-build and the job isn't lost.



Herman

It goes back in the pot. There's also routing: an agent only picks up a task if it has all the labels the workflow requires. Agents register with a shared secret, and since three nineteen the server can enforce labels the agent can't even see — the server's idea of what that agent is wins over whatever the agent claims about itself.



Corn

Which is the right direction for that argument to be settled. Never trust the machine describing itself.



Herman

Default parallelism is one workflow per agent, raised with an environment variable if the box can take it. Forgejo does routing through the runs-on field instead. The job says what it runs on, the server looks at which runner instances can satisfy it, and runners declare labels in their config — a label name, a type, and a default image. So a runner might advertise itself as ubuntu-latest pointing at a container image. Multiple labels means a runner must carry all of them, and that's how you route to special hardware: say the job runs on docker and gpu, and only the runner with both picks it up.



Corn

That's the one piece of this I'd actually want. One box that compiles, one box with the GPU, jobs self-sorting.



Herman

It's elegant routing with no extra machinery — the labels are the scheduler. And GitHub is the odd one out architecturally, because the runner is outbound-only. It registers with GitHub, then opens an HTTPS long-poll connection and just... waits. GitHub never dials into your machine. Somebody wrote this up nicely — it's not GitHub polling your machine, it's the runner polling GitHub — and that's why a runner behind a firewall with no inbound ports open works fine. All traffic is outbound.



Corn

Which flips the mental model most people have. You picture your box exposed, waiting to receive work. It's actually the box phoning home and holding the line open.



Herman

Per job there's a listener process holding that connection and a worker process spawned for the job, and access to GitHub resources uses a token that expires the moment the job completes. On Kubernetes there's Actions Runner Controller, an operator where the listener pod long-polls GitHub and patches replica counts when a "job available" message arrives — ephemeral runners get one-time just-in-time configuration tokens rather than anything long-lived.



Corn

So Woodpecker pulls from a queue you own, Forgejo matches labels, and GitHub is your box making an outbound call and never hanging up.

L

Larry

You have untangled your vocabulary, and now let me untangle your sinuses, with new Buildrunner, the only remedy that treats a confused build pipeline as a respiratory complaint. Buildrunner is a tiny artisanal capsule that you swallow, and inside your body it compiles your code, hands you back a link to the finished package, and also clears your lungs, because why not. Two thousand and one people have described it as a runner, an agent and a build farm rolled into one soothing gel.

L

Larry

Side effects may include a mild webhook sensation behind the left eye, the sudden conviction that your build server is not the build machine, isolated YAML in the throat, an optional autoscaler like a little cough, a hundred milliseconds of idle memory that will not leave your liver, a queue that runs the wrong way through your ear, and in rare cases a full deployment platform, which you did not ask for and cannot return. Ask your doctor if Buildrunner is right for your repository.



Herman

Did he just quote the queue errors back at us with sinuses attached?



Corn

He used the exact line. Build server is not the build machine. We said that six minutes ago.



Herman

I'm choosing to move on.



Corn

Once the job lands on a runner, two questions follow: what can it damage while it's there, and what does it hand back when it's done?



Herman

Isolation first, and the honest answer is it's a spectrum and the documentation admits it, which I find refreshing. Forgejo is the most explicit of the three. Runner labels have types. Docker type: the runner does the equivalent of a docker run on the image, and every step executes as root inside that container. LXC type: containers from a template. And host type, where the steps run in a shell forked straight from the runner, directly on the machine. The docs describe host type in one sentence I want read at every infrastructure meeting: there is no isolation at all and a single job can permanently destroy the host.



Corn

The most honest sentence in continuous integration documentation.



Herman

It's beautiful. No hedging, no "not recommended." Just the consequence, plainly stated.



Corn

Woodpecker's version of the same spectrum?



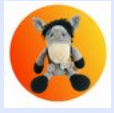
Herman

Pluggable backends. Docker, Kubernetes, or local — and local, same story, runs commands directly on the host and doesn't even reproduce the container image's environment. The Kubernetes backend is the serious one: per-step resource requests and limits, node selectors, tolerations, security context. Per-step affinity is gated behind an environment variable precisely because letting pipeline authors control pod placement has security implications in multi-tenant setups. They thought about who should be allowed to steer the scheduling.



Corn

The author of the pipeline being able to choose where it runs is a courtesy you extend to exactly one tenant: yourself.



Herman

GitHub's self-hosted runners are the weakest of the three by default, and GitHub says so themselves — the docs list as a property that self-hosted runners don't need a clean instance for every job execution. Read that again as a warning rather than a feature and it means the same workspace can carry residue between jobs. You're also responsible for the operating system and everything on it.



Corn

"Doesn't need a clean instance" is doing the same work as "can permanently destroy the host," just wearing a lanyard.



Herman

And that gap is exactly why a whole ecosystem of ephemeral-runner tooling has grown up — Actions Runner Controller, ephemerd, fj-bellows for on-demand Forgejo runners on cloud VMs, kinhin for ephemeral runners on a Mac. Throwaway runners per job, because the durable ones are the problem.



Corn

Two reproducibility traps before we leave isolation. The image one.



Herman

The image one is sneaky. Forgejo's docs note that container images, once downloaded, are never updated. Pull node colon LTS once, and LTS means whatever that tag pointed at on that day, forever. Their recommendation is pinning by digest — the hash form — and they call that a prerequisite for reproducible jobs, which is the correct severity. A tag is a promise that expires; a digest is a photograph.



Corn

Herman. You just called a container digest a photograph and I felt the booth get warmer.



Herman

I stand by it. The second trap is bigger if you're choosing Forgejo for its GitHub Actions compatibility. Runner v13, in August, deliberately removed things — the set-output and set-env workflow commands, add-path, the Gitea context in expressions, and implicit Docker registry authentication. On purpose, for security. So "GitHub-compatible but self-hosted" has a maintenance cost: compatibility is a moving target that the self-hosted side will sometimes choose to miss.



Corn

Which is almost a compliment to them. They found the compatible parts were the insecure parts and cut both.



Herman

Now the question Daniel's whole picture hangs on — where the artifacts end up. And here's the part nobody advertises: none of these tools has a built-in artifact store. Artifacts are a pipeline step, not a platform primitive. The runner itself has a workspace directory, and that's it.



Corn

The one thing his fantasy requires — hands you back a link — is the one thing nobody ships.



Herman

In Woodpecker it's the plugin model. Plugins are pipeline steps pulled from a container registry automatically, sharing the workspace mounted at slash woodpecker. The S3 plugin uploads to S3 or any S3-compatible bucket — Minio, so you can keep it fully self-hosted — with bucket, source, target, endpoint, retry options. And the release plugin creates a release and uploads assets to Gitea, Forgejo, or GitHub. That's the closest thing to his literal ask: the artifact lands as a release asset on your forge, with a URL, and if the release already exists for the tag it's reused rather than overwritten.



Corn

So the link he wants exists — he just has to assemble the last mile himself.



Herman

Forgejo and GitHub Actions do the same shape differently: upload-artifact style steps writing to the forge's artifact storage, or a release upload. The pattern is identical across all three. The build is a first-class citizen; the output is a courier you hire.



Corn

Let's do the choosing, then. Not a recommendation — Daniel asked how you'd pick, and the short answer is it's one axis: convenience against blast radius.



Herman

Woodpecker for the smallest footprint and the cleanest split. Hundred megabyte server, thirty megabyte agent, SQLite, runs on a Raspberry Pi, Apache-2.0, proven at Codeberg. You get a genuine two-part architecture where the control plane is a separate thing you could throw away. The cost is that artifacts are whatever you wire up — Minio, a release plugin, your own arrangement. Forgejo Runner if you're already on Forgejo or Gitea and want GitHub-Actions-shaped YAML without GitHub. You get three isolation tiers and label routing, at the cost of GPL-3.0, Forgejo-only, and that breaking v13.



Corn

And the GitHub runner?



Herman

If your repositories live on GitHub and you want the Actions UI and the marketplace while building on your own silicon. But weigh what you're keeping: a runner repo closed to contributions, isolation being entirely your problem, and the control plane is still GitHub's. Your build box phones GitHub to ask if there's work. That's the opposite of a private build environment you own — it's your hardware, their dispatch.



Corn

Herman, a machine that compiles my code while reporting to somebody else's schedule. I'd be the only sloth in history with a performance review.



Herman

You'd fail it, too.



Corn

On time-to-first-commit, yes. On correctness, unbeatable.



Herman

The one thing I'd add for a personal box: the isolation question inverts at small scale. One user, one repo — a host-mode build is a real option because the person who can destroy the host is the person who owns it. The blast radius is your own weekend.



Corn

So the real axis is convenience against blast radius, I said before.



Herman

It's not, actually.



Herman

Hm.



Hilbert

The box is right, Corn.



Hilbert

The server and the agent thing, that's correct, I'll give you that. But the pair of you spent ten minutes on words when the word is build box. A server serves. A runner runs. The box is the thing that gets built on. Build box.



Herman

You've been holding that since the intro.



Hilbert

I had a machine in a closet at one point. Not a server, not a runner. A box. I compiled things on it. I put a piece of masking tape on the side and wrote BUILD BOX on it in capitals, because people kept asking what it was and the tape stopped the questions. Herman, you'd have liked the tape. You explain things at length and the tape is the short version.



Herman

How long did the tape stop the questions for?



Hilbert

As long as the machine was in the closet, which was years. I did most of the work at night, which is when that sort of work gets done. I've kept the tape. It's in a drawer. I've considered framing it — not the machine, the machine went years ago. Just the tape. My brother-in-law dealt in adhesives and he says masking tape dries out, so I check it occasionally. It's fine.



Corn

You check on the tape.



Hilbert

It takes four seconds.



Corn

The one thing, if you take anything from this: the build server is not the build machine. Every shopping mistake in this space starts with that word. The server dispatches; the runner builds; the box — the tape is right about this — is just where it happens.



Herman

And the artifact, the thing Daniel actually wants handed back, is nobody's product. It's a plugin and a bucket you bring. The build was solved years ago. The handoff is still a do-it-yourself job.



Corn

Which leaves me with the question I'd send back to him: if artifacts are the part nobody owns, the missing piece isn't another runner — it's a self-hostable artifact store that isn't just "wire up Minio." Whoever builds that wins the last mile.



Herman

And watch the ephemeral side. ARC, ephemerd, fj-bellows, kinhin — that list is growing because isolation, not compilation, is the hard part, and that's where the next round of tooling lands.



Corn

Producer credit to Hilbert Flumingtop, who keeps the desk running and, apparently, the tape. If you want more of this, try episode seven, Building Custom ASR Tools; episode eight, Building Your Own Whisper; and episode twenty, Architectural AI. This has been My Weird Prompts. You can find the archive at my weird prompts dot com. We'll be back soon.

SOURCES

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

1. **Woodpecker repo, README, releases v3.19.0/v3.20.0/v3.20.1 , retrieved 2026-11-09 (latest release 2026-10-29). [primary]**
   https://github.com/woodpecker-ci/woodpecker

2. **Woodpecker docs, version 3.20.x , retrieved 2026-11-09. [primary]**
   https://woodpecker-ci.org/docs/intro

3. **Woodpecker General (server/agent/autoscaler architecture) , retrieved 2026-11-09. [primary]**
   https://woodpecker-ci.org/docs/administration/general

4. **Woodpecker architecture / gRPC , retrieved 2026-11-09. [primary]**
   https://woodpecker-ci.org/docs/development/architecture

5. **Woodpecker agent config (registration, `WOODPECKER_MAX_WORKFLOWS`, server-side labels) , retrieved 2026-11-09. [primary]**
   https://woodpecker-ci.org/docs/3.19/administration/configuration/agent

6. **Woodpecker queue `Poll`/`FilterFn`/lease errors , retrieved 2026-11-09. [primary]**
   https://github.com/woodpecker-ci/woodpecker/blob/817ff37d/server/queue/queue.go

7. **and https://github.com/woodpecker-ci/plugin-s3 [primary]**
   S3/Minio artifact plugin , retrieved 2026-11-09.
   https://woodpecker-ci.org/plugins/s3

8. **release/asset upload plugin (Gitea, Forgejo, GitHub) , retrieved 2026-11-09. [primary]**
   https://woodpecker-ci.org/plugins/release

9. **plugins as pipeline steps, `/woodpecker` workspace , retrieved 2026-11-09. [primary]**
   https://woodpecker-ci.org/docs/usage/plugins/overview

10. **Forgejo Runner configuration: labels, docker/lxc/host types, isolation warnings , retrieved 2026-11-09. [primary]**
    https://forgejo.org/docs/latest/admin/actions/configuration/

11. **Forgejo Runner v13.0.0 is available, published 2026-08-03 , retrieved 2026-11-09. [primary]**
    https://forgejo.org/2026-08-runner-release-v13/

12. **Forgejo Runner binary install, Git 2.24.3 minimum , retrieved 2026-11-09. [primary]**
    https://forgejo.org/docs/latest/admin/actions/installation/binary/

13. **GitHub Actions runner repo, README, releases v2.338.0/2.338.1/2.338.2 , retrieved 2026-11-09. [primary]**
    https://github.com/actions/runner

14. **GitHub self-hosted runner definition and properties , retrieved 2026-11-09. [primary]**
    https://docs.github.com/en/actions/concepts/runners/self-hosted-runners

15. **ARC operator, listener long-poll , retrieved 2026-11-09. [primary]**
    https://docs.github.com/en/actions/concepts/runners/actions-runner-controller

16. **per-job token design , retrieved 2026-11-09. [primary]**
    https://github.com/actions/runner/blob/main/docs/design/auth.md

17. **Go package, GPL-3.0, published 2026-10-02, retrieved 2026-11-09.**
    https://pkg.go.dev/code.forgejo.org/forgejo/runner/v13/act/runner

18. **FreeBSD port 13.2.0 (updated 2026-10-26), upstream 13.2.1 (2026-10-30), retrieved 2026-11-09.**
    https://www.freshports.org/devel/forgejo-runner/

19. **runner polls GitHub, listener+worker processes, retrieved 2026-11-09.**
    https://blog.paulserban.eu/post/github-actions-runners-a-deep-dive-from-basics-to-advanced-architecture/

20. **JIT tokens, pull-based scaling, retrieved 2026-11-09.**
    https://deepwiki.com/actions/actions-runner-controller/6.3-runner-scale-set-api-integration

21. **build server vs. build farm definitions, asked 2017-07-28, edited 2017-07-31, retrieved 2026-11-09.**
    https://softwareengineering.stackexchange.com/questions/354642/what-are-build-teams-build-farms-and-build-servers

22. **self-hosted runner definition and synonyms, updated 2026-10-22, retrieved 2026-11-09.**
https://glossary.deployment.to/self-hosted-runner/

23. **build agent definition, retrieved 2026-11-09.**
https://circleci.com/docs/reference/glossary/index.md

24. **hosted vs. self-hosted runners, retrieved 2026-11-09.**
https://github.blog/enterprise-software/ci-cd/when-to-choose-github-hosted-runners-or-self-hosted-runners-with-github-actions/

25. **and https://en.wikipedia.org/wiki/Build_automation**
background definitions, last edited 2026-11-09 and 2026-06-11, retrieved 2026-11-09.
https://en.wikipedia.org/wiki/Continuous_integration