06 NOVEMBER 5, 2026 4 PP

Moving 199 repos off GitHub Actions in eight days

Felipe Arce · Tim Child · Collin Rijock

After a three-hour GitHub Actions outage, we built Roadrunner, a self-hosted Woodpecker and Dagger deploy lane. Eight days later it was the default deploy path for 199 repos, with GitHub Actions kept as the fallback.

CI-CD INFRASTRUCTURE WOODPECKER DAGGER GITHUB-ACTIONS RELIABILITY BROAD-DAYLIGHT OPEN-SOURCE



When I first onboarded to GitHub Actions I remember thinking that this was how it was meant to be done all along. The idea of having the code and the CI/CD live in the same space made so much sense and seemed like the “right way” to do things. The pricing was fantastic, the speed and dev experience were great, and everyone was happy with the system as it was for years to come.

As we all know, this has changed recently. This isn’t just another technical blog dogpiling on top of GitHub SLAs and making fun of missing 9’s; instead I want to share how we built Roadrunner for Exowatt’s software factory and reduced costs, improved deployment speeds, and weaned off the dependency we had on GitHub Actions.

On October 5, 2026, at 19:12 UTC, GitHub opened an incident for Actions, and for the next three hours and 38 minutes a merge to main at Exowatt mostly did not deploy. Two hours in I asked one question: over the last 90 days, how often had Actions actually gone down compared with GitHub's webhooks? Actions had 25 incidents, about 69 hours in total. Webhooks had 5, about 7 hours.[1]

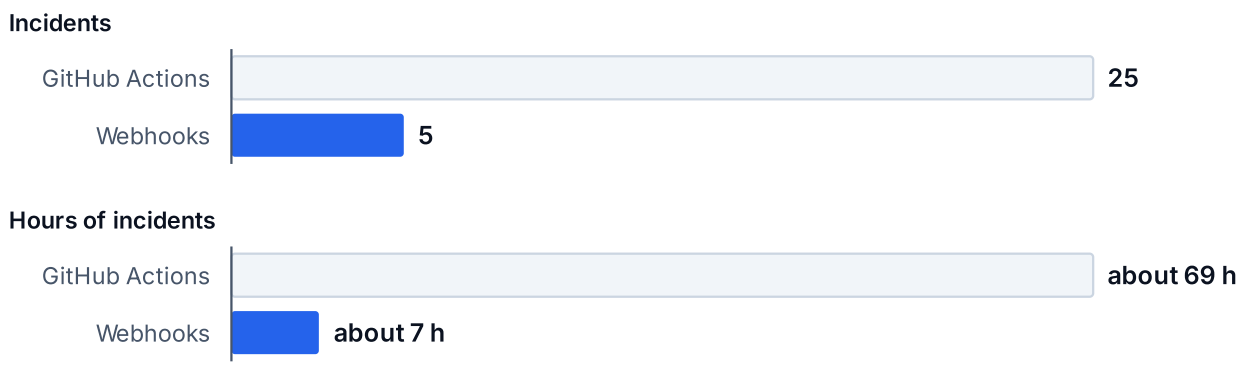


Figure 1. GitHub incidents in the 90 days to October 5, 2026, Actions against webhooks. Each panel has its own scale.

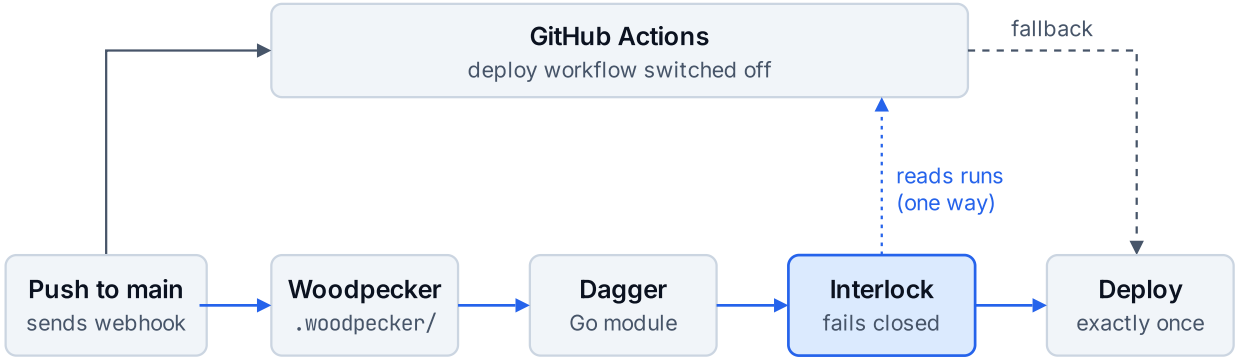
Back to webhooks

The main idea of Roadrunner was to actually go back to an old school way of doing GitHub-based CI/CD: webhooks. A webhook is just an HTTP POST, so a deploy lane that listens to webhooks and runs on our own hardware rides out most Actions outages. But unlike times in the past where this would be some slow Jenkins run or some Ansible workflow, we leveraged Dagger to make a hyper-cacheable, performant, reliable alternative that would then fuel an agentic self-improvement loop for our CI/CD jobs (read more: LMNOP - our CI/CD self-improvement agent). By owning the full stack and having total and complete control over runs, reruns, insights, etc., we unlocked the full power of an in-house CI/CD stack.

I set three rules that evening. The lane had to keep deploying during an Actions outage. It had to be open source and self-hosted, with no new SaaS to manage. And it had to run beside Actions on the same commits, so we could compare the two before trusting either one.

Those rules pretty much picked the tools: Woodpecker, an open-source CI server driven by webhooks, and Dagger, which runs pipelines as code on a cached build

engine. Woodpecker stays thin: each repo's YAML calls one function in a shared Go module that knows how to test, build and deploy (Actions can call it too). No Dagger Cloud and no Kubernetes, just a Woodpecker server in a container and two engine VMs on separate Proxmox nodes with 20 job slots, and only the webhook endpoint faces the internet.



Roadrunner: one Woodpecker server, two Dagger engine VMs on separate Proxmox nodes, 20 job slots

Figure 2. The deploy path. The interlock reads GitHub's API before every deploy; GitHub Actions never learns Roadrunner exists, and only deploys when the watcher hands a repo back to it.

Deploy exactly once

Two lanes that can both deploy will eventually both deploy, so our one hard rule is that GitHub Actions never learns Roadrunner exists. All the yielding happens on our side: an interlock in the module reads GitHub's API and only deploys if no Actions deploy is queued or running, nothing at this commit or newer has deployed, and no other Roadrunner run has claimed the commit (it claims it with a `ci/failsafe-deploy` status and checks again before rolling). Any API error is a skip.

An outage watcher decides when GitHub is down: once a minute it samples the status page, open Actions incidents and two probes of our own, and it trips after three bad samples and clears after 15 healthy minutes. Replayed against every GitHub status update since July 8, it caught 14 of 14 Actions outages a median 2.2 minutes after GitHub flagged them, and stayed quiet through 48 incidents in other GitHub services. [2]

Rolling it out

We onboarded repos in order of how often they deployed: the top 10, then 20, 50 and 100. Every repo got a drill: disable its Actions deploy, switch Roadrunner on, merge, confirm exactly one deploy, restore. Agents ran the waves in parallel under one rule (pass the drill or get reverted), and by October 8 the watcher covered 170 repos.

That night we forced the failsafe on for all 170 for two hours with every Actions deploy workflow disabled, and Roadrunner deployed all 16 commits that landed: merge to healthy in 2.3 minutes at the median, no double deploys, at most 4 of 20 slots busy.[3] Then the lab network went down, took the re-enable job with it, and 97 repos sat with Actions deploys off for almost four hours (not our finest moment), so now the watcher restores every workflow it touches by itself.

On October 9 we flipped it around, with Roadrunner as the default and Actions as the fallback. If a push has no Roadrunner deploy after 10 minutes, or the deploy fails, the watcher re-enables and dispatches the Actions workflow and hands the repo back 15 minutes later. The canary was 17 low-risk repos, and 167 had moved by evening. On October 13 we added fleet failover: if Roadrunner fails every health check for five minutes, every repo goes back to Actions at once.

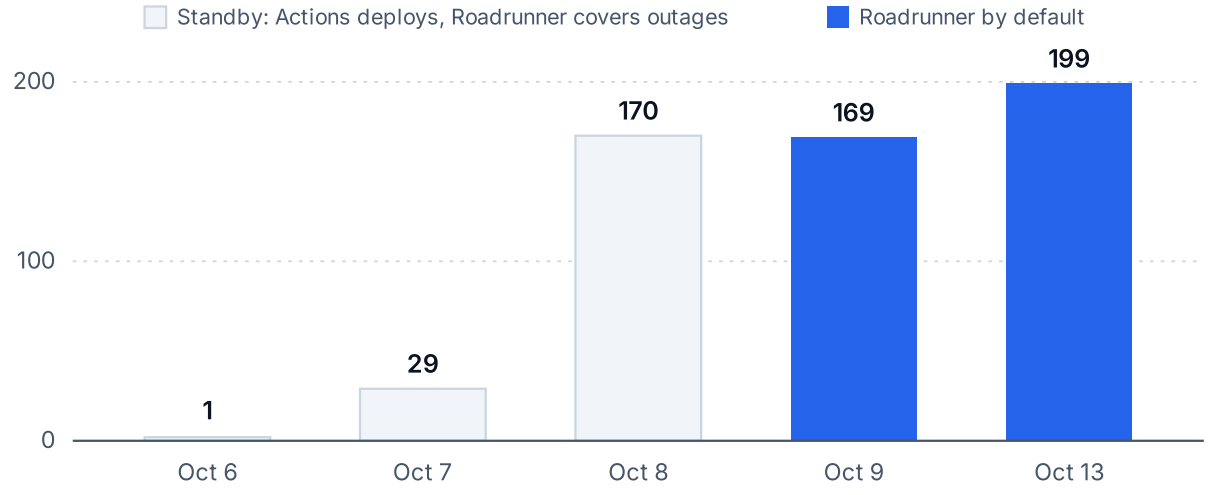


Figure 3. Repos with a Roadrunner deploy lane, by day. The default flipped on October 9; two more repos stayed on standby.

In its first 80 hours as the default, 10 deploys fell back to Actions, all Roadrunner missing its own window, and each repo came back in 19 minutes on average. Our agents read build status with a `gh rr` CLI extension instead of polling Actions.

Old vs new

Eight days after that outage, Roadrunner was the default deploy path for 199 repos, and only three repos that had deployed in the last 90 days still went through Actions alone. Of those 25 Actions incidents, 14 actually stalled or failed runs (about 48 hours in all), and for a Roadrunner repo that now changes nothing.

It is also a lot faster. In the week to October 13, across 3,403 pipelines, the CI stage took 25 seconds at the median and 262 at p95; the same repos took 131 and 1,017 seconds on Actions.[4] Queue wait was 4 seconds, where an audit three days before the outage had our self-hosted Actions jobs waiting about 232 hours a week for one pool of three runners.[5] And about 35,000 runner-minutes a week left GitHub for our own hardware.

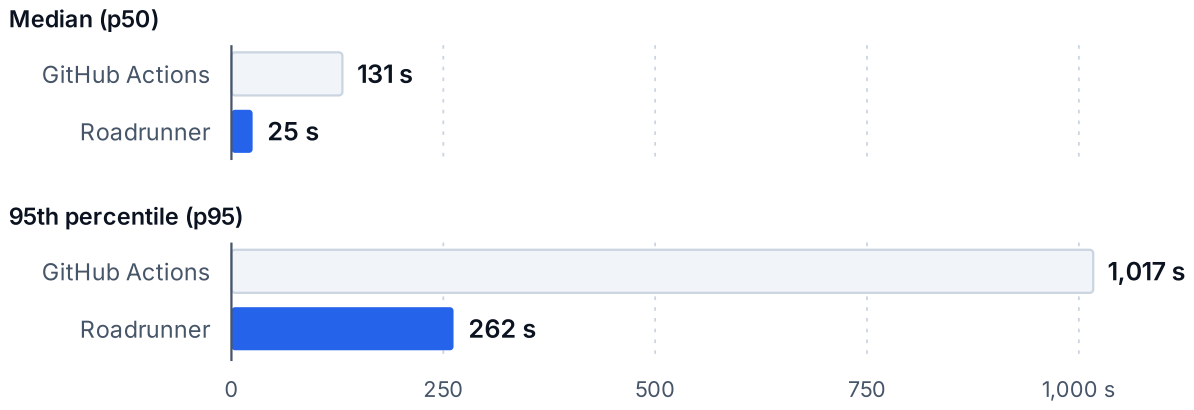


Figure 4. CI stage time for the same repos, GitHub Actions against Roadrunner (week to October 13, 3,403 Roadrunner pipelines). One scale, in seconds.

The first failsafe deploy ran 3 hours and 21 minutes after we wrote down the requirements, and the plan had estimated two and a half weeks.

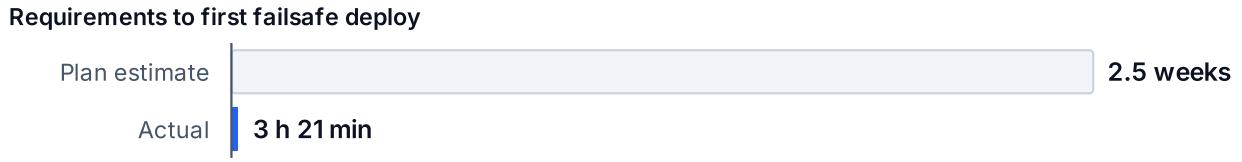


Figure 5. Planned against actual time from written requirements to the first failsafe deploy, on one scale.

	GitHub Actions (before)	Roadrunner (after)
CI stage, median	131 s	25 s
CI stage, p95	1,017 s	262 s
Queue	about 232 h a week of waiting for one pool of three runners	4 s wait at the median
Self-hosted job capacity	three runners in the pool we queued on	20 job slots on two engine VMs
Actions outages that stalled runs (14 in 90 days, about 48 h)	deploys stall	nothing changes
Runner-minutes on GitHub	about 35,000 a week	moved to our own hardware
Requirements to first failsafe deploy	2.5 weeks (plan estimate)	3 h 21 min

Next: proving local builds

CI still repeats, cold, the builds and tests developers already ran before opening a PR. MBS, the Merkle Build Service we designed in late October, turns every compiler call, check and test into a node keyed by the hash of its inputs, and the server re-runs only what its Merkle ledger has never seen. I first pushed for trusting the developer’s signature on test results and withdrew that the same day: a signature proves who ran the tests, not that they passed. On one Rust PR so far, push to green took 15 minutes 43 seconds against 22 to 25 on Actions.[6]

Roadrunner now deploys almost every Exowatt app, and GitHub Actions is the fallback we keep for the day Roadrunner breaks. It took eight days and no new

vendor, and we are open-sourcing it under Broad Daylight, Exowatt's open-source initiative.

GitHub Status incident history for the GitHub Actions and Webhooks components, 90 days to October 5, 2026: githubstatus.com/history.

Roadrunner outage-watcher backtest against every GitHub status update from July 8 to October 6, 2026 (Exowatt, internal).

Roadrunner failsafe two-hour exercise report, October 8, 2026 (Exowatt, internal).

Roadrunner pipeline numbers for the week to October 13, 2026, against GitHub Actions run history for the same repos (Exowatt, internal).

Exowatt CI audit of self-hosted GitHub Actions runner queues, October 2, 2026 (internal).

MBS measured runs, second push of one Rust PR, Exowatt cicd documentation (internal).