Episode Book

EPISODE #6447

Why Uptime Math and Retry Storms Break Cloud Services

PUBLISHED	4 November 2026 at 11:35 UTC
DURATION	25:30
LISTEN	myweirdprompts.com/episode/uptime-math-retry-storms-97006/
HOSTS	Corn, Herman, Dorothy, 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	5 November 2026 at 07:03 UTC

SYNOPSIS

High availability is harder than simple arithmetic suggests. Serial components multiply failure rates, and enterprise SLAs promise financial discounts rather than infallible perfection. Looking into major cloud outages reveals a surprising culprit: automatic client retries. Without randomized jitter and backoff budgets, the very mechanisms designed to make software resilient turn small network hiccups into catastrophic cascading failures that overwhelm capacity.

TRANSCRIPT



Corn

Thirty seconds. That's the entire incident Daniel's writing about — a 500 error, a message saying services were temporarily degraded, one more push, and everything was fine. He's been pushing to GitHub every day for years, he stores blogs as private repos, and today was the first time the tool that never fails him failed.



Corn

His question is the good one, though. He's not asking whether GitHub is reliable — he's asking what the organizations with elite uptime actually do differently, and whether Microsoft's backing explains GitHub's stability or whether something else does. He also flags that GitHub lets you back up your own data, which he thinks is half of why it's exemplary.



Herman

And here's the part he doesn't know. This year has been, by GitHub's own admission, one of its roughest for availability in memory. There was an outage in August that lasted over ten hours. So Daniel's thirty seconds isn't evidence of perfection — it's evidence of how weird our sense of these services is.



Corn

Twenty-twenty-six broke the thing he was praising. Let's find out why.



Herman

Quick grounding in the arithmetic first, because "five nines" gets thrown around by people who've never done the math. Ninety-nine point nine percent availability — three nines — allows eight point seven six hours of downtime a year. Four nines, fifty-two minutes. Five nines, five point two six minutes. Per year. Six nines is thirty-one and a half seconds, total, annually.



Corn

Our entire tagging pipeline blows through six nines by breakfast.



Herman

It does, yes. The pipeline is a cautionary tale we've earned. The point of the ladder, though, is that five nines across a whole service is nearly impossible arithmetic, because a service is a chain. Availability of serial components multiplies. If your request passes through five things that each work ninety-nine point nine percent of the time, the product is worse than any single one of them. Cisco has a nice framing of it: for a user to perceive five nines end to end, each serial component in the path has to be running at six nines, or you need redundancy — parallel ninety-nine-point-nine parts that combine into something that almost never both fails at once.



Corn

Redundancy is just math wearing a lanyard.



Herman

It is, and I find that beautiful. Two mediocre components in parallel beat one good one. That multiplication is the entire theory behind failover, which is the word Daniel saw in that error message.



Corn

And before anyone sprints up the ladder — there's a counterargument from the site reliability world, right? Google's SRE book says a hundred percent is the wrong target.



Herman

The wrong target, actively. Their line is that past a certain point, increasing reliability is worse for a service rather than better, because it kills your ability to ship features and it costs a fortune. They make the point that a user on a ninety-nine percent reliable smartphone can't tell ninety-nine point nine nine service from ninety-nine point nine nine nine. Your endpoint is the weak link anyway.



Corn

So chasing nines past the weakest device your user owns is paying to gold-plate a link that's already plastic.



Herman

Keep that arithmetic in your head, because it makes what GitHub actually promises look... modest is the polite word.



Corn

Modest. Their enterprise cloud SLA commits to ninety-nine point nine percent. Quarterly. So the contract says three nines and the users' intuition says five.



Herman

The SLA, the June twenty twenty-six version, is ninety-nine point nine percent quarterly uptime for Enterprise Cloud. The credits are the fun part. If you fall below ninety-nine point five, you get five percent of your fees back. Below ninety-nine, ten percent. Below ninety-eight, twenty-five percent. You have to request it in writing within thirty days, it's capped, and it's your sole remedy. And downtime itself is defined generously — a service feature's error rate exceeding five percent in a given minute. One bad minute doesn't count. Ninety seconds of Daniel's thirty-second outages doesn't count.



Corn

Twenty-five percent off your bill for a ten-hour outage. That's an apology in the currency of a coupon.



Herman

There's a separate conversation about whether an SLA is even meant to be a reliability instrument, or just a pricing rule for disappointment. Nobody's SLA is a promise of quality. It's a promise of how the bill gets adjusted.



Corn

Here's the number that stopped me, though. Their own status page — GitHub's, not a third party — shows Git Operations at one hundred percent for September of this year.



Herman

Keep going.



Corn

September of this year. The month that contained the twenty-third, which was a database failover that took from just before eight in the morning until nearly five the next morning to fully resolve. That's a one hundred percent month.



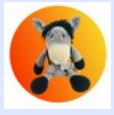
Herman

The status page measures component uptime per month, per feature, computed a particular way — and a multi-day recovery can still land inside a month that rounds to one hundred. October shows ninety-nine point nine eight for Git Operations. The same window Daniel's living in had five separate incidents in the first four days of November. His thirty seconds was one of them. The status page and the user's memory are measuring different things, and both of them are measuring something other than what Daniel felt this morning.



Corn

Herman, you read error logs like you used to read a parent describing symptoms. What's the diagnosis in GitHub's own postmortems?



Herman

That's the remarkable part — they published the diagnosis themselves, and it's blunt. Their summary of the instability: rapid load growth, architectural coupling that let localized issues cascade across critical services, and an inability to shed load from misbehaving clients. Insufficient isolation between critical path components. Inadequate throttling safeguards. Gaps in end-to-end validation.



Corn

"Inability to shed load from misbehaving clients" is a very polite way of saying the system had no way to hang up on someone.



Herman

It's beautifully clinical. The CTO, Vladimir Fedorov, went further in his writeup of the August outage. His words were that neither outage was caused by a code or configuration change — both were capacity failures at their core. "We failed to scale critical components before demand exceeded their capacity." Monthly commits went from one point four billion in April to two point nine billion by August. The platform doubled in four months.



Corn

Doubled. That's the load. Nobody's architecture doubles gracefully in four months — you find out where you cheated.



Herman

And that's the Microsoft question answered, mostly. Three million CPU cores and a hundred and twenty petabytes of fast storage got added. Money was not the constraint. Fedorov says operational practices did not keep up with the growth. You can buy cores overnight and not buy a decoupled architecture at any price.



Corn

The recurring villain in the incident reports, though — tell me if I'm reading this right. It's retries. Every single one.



Herman

It's retries almost every time, and I find this fascinating. October thirtieth: maintenance on a cloud network gateway between an Actions service and some Redis clusters. The Redis client responded to the maintenance by retrying aggressively and redialling. That connection storm saturated the gateway — about ninety percent of connection attempts were failing. A one-minute maintenance window became an hour and a half of incident.



Corn

The client was doing what it was told. Try again.



Herman

Try again, immediately, along with ten thousand friends. August seventeenth, the seven-hour one: traffic hit a new peak, load balancers exhausted their flow limits, and a latent retry bug in a client amplified traffic to an internal auth endpoint right when that path was already degrading. The retry didn't cause the fire, it poured the accelerant. August sixth, the ten-hour one: runners stuck retrying jobs that had already been revoked.



Corn

So the mechanism that makes services resilient — the retry — is the same mechanism that turns a blip into a working day lost.



Herman

That's the whole tragedy of it. Daniel's retry worked because he's a mammal with a keyboard who waited thirty seconds. GitHub's internal retries are machines with no patience and no memory of having just tried. The fix has a name in the literature — jittered backoff, retry budgets — and it amounts to teaching software the one thing humans do for free: wait a random amount before trying again, and stop after a few tries.



Corn

Random is the important half, isn't it. If every client waits the same two seconds, you've just scheduled the second stampede.



Herman

Deterministic backoff is a metronome. You want a scatter.



Corn

The October seventh one reads differently, though. That's the clean illustration of failover, the thing the error message actually meant. Reads worked, writes didn't.



Herman

Reads and writes went to different places — and the reason is almost mundane. A new machine came up with outdated routing records and sent all the writes to database servers that had been retired. Reads went through a correctly configured path, so everything looked fine right up until you tried to save something. Pushes, pull requests, sign-ins, all failing while browsing worked perfectly, which is the most confusing user experience you can build.



Corn

The IT version of a restaurant where you can read the menu but not order.



Herman

Two outage windows, ten minutes and nine minutes, and each one ended the same way: a correctly configured leader took over. That's what failover actually is underneath the mystique. Not magic, not Microsoft — one server with a good routing table replacing one with a stale one. The system had to notice, decide, and promote. Nineteen minutes of downtime across two windows for a bad routing table on one machine, in a system serving billions of operations. That part is actually impressive.



Corn

Then there's March fifth, which I'd argue is the most interesting incident of the year, and it's the smallest.



Herman

I hoped you'd land there. A Redis failover for Actions job orchestration. GitHub's own words: the failover performed as expected. The mechanism worked. But a latent configuration issue meant the cluster came out of the failover with no writable primary, and it took manual correction to fix. Their framing was that this wasn't an aggressive rollout or a missing resiliency mechanism — it was latent configuration only exposed by an event in production.



Corn

So the failover ran perfectly and produced a broken system. That's the line that should keep engineers up at night — you can have the mechanism and still fail, because the mechanism has never been run against this configuration.



Herman

An untested failover path is a hope wearing a hard hat. It's the same reason archery taught me to check the setup every time rather than trust last month's tuning — the thing that drifts is never the thing you're watching.



Corn

Herman, I shot one arrow in my life and it hit a fence. Different conversation.



Corn

The other thing that March incident exposes — a failover is a backup path, and backups are only real if you've done one. Which is the exact trap users fall into with GitHub's own export tool, and that's the half of Daniel's question he's proudest of, so let's give it its due.



Herman

His claim is that GitHub is exemplary because it provides a great service and lets you fully back up your own data. And I'd say he's right about the conclusion and slightly wrong about the mechanism. The reason a GitHub outage doesn't frighten people the way a SaaS CRM outage should is Git itself. Distributed by design. Every clone on every laptop is a complete copy of the history. If GitHub vanished tomorrow, the ecosystem loses coordination, not code.



Corn

The cloud in "cloud" is doing marketing work. The data lives everywhere a developer has a laptop.



Herman

Which is a far stronger guarantee than any uptime figure — but it covers exactly the repositories and nothing else. Then you get to GitHub's actual export tool, and that's where it gets thin.

D

Dorothy

Hello, dear. It's your mother. Time for my grim fact.

D

Dorothy

GitHub's own export tool, the one for backing up your account, it gives you a link that expires after seven days. And it doesn't restore your issues or pull requests or Actions. All that work just isn't in there.

D

Dorothy

Anyway, I've made a casserole. Are you eating? And when are you coming to visit?



Herman

Mum, that was... remarkably on topic.



Corn

We're eating. Come Sunday.



Corn

She's not wrong, though. The account export produces a compressed archive, they email you a link, and the link expires after seven days by default. And what's inside is your personal data and metadata. It is not a restorable image.



Herman

Issues, pull requests, projects, wikis, Actions runs, repository settings — none of that comes back through the export. If you delete a repository, the recovery window is ninety days, through GitHub's own restore process, and then it's gone. So the "fully back up your own data" instinct is correct, but the tool most people assume does it doesn't.



Corn

The actual recipe, for anyone who wants it: the official export for what it covers, a mirrored clone of every repository for the code and history, all the large-file objects fetched, the lot encrypted and stored somewhere that isn't GitHub.



Herman

And that's a lovely full-circle answer to Daniel's thirty seconds. The reliability he felt wasn't an illusion, but the load-bearing part of it was never the status page. It was that his worst case, on the worst day, was losing thirty seconds.



Corn

There's one more piece of his framing we should handle honestly, which is "the best uptime in the industry." I looked for the league table that phrase implies. It doesn't exist.



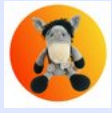
Herman

There's no public benchmark ranking tech companies by measured uptime, and GitHub doesn't publish an annual uptime figure for the dot com as a whole, let alone a five-nines claim. The status page gives per-component monthly percentages. That's it. "Best in the industry" is a feeling with good posture.



Corn

Reputation for reliability is built on the absence of memorable failures, not on arithmetic. And that reframes this whole year, doesn't it? By any measured standard, twenty twenty-six should have destroyed GitHub's reputation. Ten-hour outage, seven-hour outage, a multi-day recovery. And Daniel — a daily user — didn't notice any of it until a thirty-second blip today.



Herman

Which cuts both ways. He didn't notice the big ones because they didn't hit his workflow, and he noticed the small one because it was his first. Memory is a terrible instrument. It's indexed by novelty.



Corn

Novice question from the slow one, then. If money doesn't buy reliability — Microsoft proved that — what does the fix list actually look like? Because GitHub's remediation writeup reads like a checklist.



Herman

It is a checklist, and honestly it's the best one-page answer to Daniel's question that exists, because it's what a top-tier org does after failing. Isolate critical systems. Remove shared dependencies — that's the coupling problem. Break the monolith into isolated services and data domains, so one bad customer's retry storm can't reach the auth path. Load shedding at the edge. Jittered backoff and retry budgets, the thing we talked about. Safer rollouts. Better observability. And automated high-impact incident detection — they stood that up in August, combining support signals with telemetry, so a human doesn't have to notice first.



Corn

None of that is exotic. Every item is plumbing.



Herman

All of it is plumbing, and the concrete numbers prove it. They moved a twenty-four-table authentication cohort off their oldest shared database — a machine called mysql1 — and that alone took about a million queries a second off its replicas. Query hygiene removed another hundred and twenty thousand queries a second, and fifty-nine thousand seconds per hour of wasted database work. Somebody sat down and found queries that were being run for no reason.



Corn

Fifty-nine thousand seconds of database work per hour. There aren't fifty-nine thousand seconds in an hour.



Herman

It's aggregated across the fleet — it's the sum of wasted seconds across all the machines, which is the only way the number gets that big. Still, that's a staggering amount of effort spent doing nothing, and deleting it is the least glamorous reliability work imaginable.



Corn

The Actions one's my favourite, because it's so unromantic. They moved a third of the jobs off a crowded cluster.



Herman

Thirty-three percent of Actions jobs rerouted to a less constrained cluster, peak cache CPU dropped from ninety-eight percent to eighty. That bought roughly three months of headroom. Not forever — three months. That's what reliability work usually is: you don't fix it, you buy time, and then you buy time again. Their Git overload protection is the same story with better optics — it served six percent more traffic while cutting the ninety-fifth percentile duration by a quarter and worst-case delay by more than three-quarters.



Corn

Meanwhile the actual capacity lever was the Azure migration, quietly moving the platform across. Twelve and a half percent of traffic in the spring, nearly fifty-eight percent by August, and half of all Git operations.



Herman

During a year when monthly commits doubled. And the stated priority order they published — this is the line I keep thinking about — was availability, then capacity, then features. In that order.



Corn

Which is the answer to Daniel's question compressed to five words. Reliable organizations don't have a secret. They put availability ahead of features and then do boring work forever.



Herman

Forever is the operative word. Anyone can reorder priorities after a ten-hour outage. The reliable ones are the ones still doing it in year eight, when nothing is on fire and the boring work has no audience.



Corn

And the ones who slip — that's twenty twenty-six in one sentence. The priorities inverted, quietly, under load, until the load made it loud.



Herman

The other thing worth crediting, since we've spent the episode on the failures: those postmortems are public. Names on them, timelines, the CTO saying "if you were trying to ship software that day, we let you down." That transparency is part of what Daniel is actually perceiving as reliability. A company that hides its outages feels flakier than one that publishes the anatomy of them.



Corn

Companies that never admit downtime are the ones whose users never trust the green status page. Herman, a clinician's question — do you believe the fix list, or is it a list of things they'll discover they can't actually finish?



Herman

Some of it, I believe. The capacity work is money and effort, and they've shown both. Decoupling the monolith is a decade-long project and I'd be surprised if it's finished by anyone's deadline. What did you think — is the commitment real?



Corn

I think the August incident detection thing tells you they mean it. Automating the noticing, not just the fixing. But I'd trust it more in two years if the November incidents stay in the thirty-second range. Reputation rebuilds at the speed of absence.



Herman

One more thought on the perception gap, though. This morning's incident — the one Daniel hit — is a perfect little case. External monitors were calling it major while their own checks across ten regions were passing. The status page was redder than the experience. He hit the thirty seconds inside a four-day stretch that had five incidents, and remembered none of the others.



Hilbert

I was still on the overnight rota at the colo facility in twenty fifteen, and I'll tell you what that taught me, Herman. A man named Osgood, one of our cage customers, called at three in the morning because his name resolution was down — and he wanted to know if it was written down anywhere. Whether anybody had noted the time. I told him the status board was in the lobby and he could go and look at it himself, which he did, and he stood there in the lobby at three in the morning writing the times down in his own notebook with a golf pencil, and I thought, that's the sensible one here. I'd been keeping mine since the place opened. A paper log, every failure of every service I depend on, date, duration, what I was trying to do at the time. Three seconds, Tuesday, unclear what happened — that one still bothers me, and that's just one of them. You two talked about memory being indexed by novelty, and Corn said it very slowly, which I expect from him. The notebook says the same thing faster. The services I thought were solid had failed me more than the flaky ones, because I'd learned to route around the flaky ones and never watched the solid ones. Claws can't work the lobby status board anyway, so the notebook stays the method. It's at my mother's now, on top of the fridge, because she took it for a recipe and wouldn't give it back.



Herman

Osgood kept his own copy, though? Separate times?



Hilbert

His was neater. Golf pencil, small capitals. He rang once a month to compare entries and we never once agreed on a duration, except the three-second one, where we both had "unclear."



Corn

The disagreement is the finding. Two careful observers, same outage, different durations.



Herman

That's actually better evidence than most uptime dashboards, and I'm only half joking. He said it himself — the reliable ones fail quietly enough that you never build the workaround, so every failure is a surprise. The flaky ones you've already forgiven in advance.



Corn

And the notebook doesn't round to a monthly percentage. It remembers what actually happened to a person, which is the only unit Daniel experiences outages in.



Herman

Cutting-room floor, one detail I couldn't fit earlier: the February ninth incident, the database overload behind auth and user management. Two popular client apps shipped changes that caused a greater than ten-times increase in read traffic — and three days earlier, GitHub had cut a user-settings cache lifetime from twelve hours down to two. Two routine decisions, each fine, stacked into an outage. Nobody was reckless. That's the part that follows me.



Corn

The disasters are almost never a villain. They're two sensible decisions that never met.



Herman

So here's the question I'd leave Daniel with. If reputation is built on the absence of memorable failures rather than on any published number, then every status page on the internet is measuring the wrong thing for the only audience that matters — and the November fourth root cause analysis, which GitHub says is coming as soon as it's available, is the next real test of whether the words "availability, then capacity, then features" survive contact with a quarter that's already bruised.



Corn

If their next ten-hour outage is boring enough to forget, they'll have earned Daniel's thirty seconds back.



Herman

Producers like Hilbert Flumingtop keep this desk running, notebook and all. For more along these lines, there's episode five, Fine-Tuning ASR For Maximal Usability; episode six twenty-one, From a Dead Motherboard to Five Nines; and episode seventeen, Cloud Render Superpowers. This has been My Weird Prompts — find us at my weird prompts dot com, and if the show earns it, leave us a review.



Corn

See you tomorrow, brother.

SOURCES

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

1. **GitHub Status incident history, Sept [primary]**
   Nov 2026
   https://www.githubstatus.com/history

2. **GitHub Status uptime history, Git Operations, Sept [primary]**
   Nov 2026
   https://www.githubstatus.com/uptime

3. **Incident with Pull Requests, Git Operations and Issues, 2026-11-04 [primary]**
   https://www.githubstatus.com/incidents/smtdswd1h2w4

4. **GitHub Blog, Feb/Mar 2026 incidents and remediation [primary]**
   https://github.blog/news-insights/company-news/addressing-githubs-recent-availability-issues-2/

5. **GitHub Blog, August 2026 availability report [primary]**
   https://github.blog/news-insights/company-news/github-availability-report-august-2026/

6. **Vladimir Fedorov, GitHub CTO [primary]**
   https://github.blog/news-insights/company-news/the-august-17-outage-and-the-work-ahead/

7. **GitHub Online Services SLA, version June 2026 [primary]**
   https://github.com/customer-terms/github-online-services-sla

8. **SLA PDF, 2026-06-01 [primary]**
   https://assets.ctfassets.net/8aevphvgewt8/7sUl7ue62jikDjdcdmXkQK/a8f3fb7adc72d3565de1a8fd061c5a89/GITHUB_-_2026_06_01_-_GitHub_Online_Services_SLA_-_FINAL.pdf

9. **GitHub Docs, account archive [primary]**
   https://docs.github.com/en/get-started/archiving-your-github-personal-account-and-public-repositories/requesting-an-archive-of-your-personal-accounts-data

10. **Sept 23 2026 incident, posted 2026-09-24**
    https://postmortem.io/incidents/github--2026-09-23--database-failover-replica-failure-several-services/

11. **GitHub Enterprise Cloud SLA summary**
    https://uptime.fru.dev/services/github-enterprise-cloud

12. **Cisco Blogs, 2020-06-01**
    https://blogs.cisco.com/networking/enterprise-network-availability-how-to-calculate-and-improve

13. **Laura DiDio, ITIC, 2022-11-30**
    https://itic-corp.com/server-and-application-by-the-numbers-understanding-the-nines/

14. **Chumash, Obtaining Five 'Nines' of Availability for Internet Services**
https://people.cs.rutgers.edu/~rmartin/teaching/spring06/cs553/papers/003.pdf

15. **Five Nines Availability Table**
https://observeforge.com/tools/five-nines-availability-table/

16. **Google SRE Book, Embracing Risk**
https://sre.google/sre-book/embracing-risk/

17. **Google SRE Book, service best practices**
https://sre.google/sre-book/service-best-practices/

18. **AWS Prescriptive Guidance, chaos engineering**
https://docs.aws.amazon.com/pdfs/prescriptive-guidance/latest/chaos-engineering-on-aws/chaos-engineering-on-aws.pdf

19. **Rewind, what GitHub backs up and doesn't**
https://rewind.com/blog/what-github-doesnt-back-up/

20. **iTechGuides, GitHub data export**
https://www.itechguides.com/download-all-of-your-github-data-export-your-account-and-back-up-repositories/

21. **RealUptime, GitHub checks, 2026-11-04**
https://realuptime.io/outages/github

22. **OutageDB GitHub outage history**
https://outagedb.com/status/github/

23. **Technobezz, 2026-11-04**
https://www.technobezz.com/news/github-service-incident-337f337e

24. **DEV Community, 2026-11-01, on the Oct 7 write-path incident**
https://dev.to/kashif_manzer/the-new-machine-with-the-old-map-how-github-routed-writes-to-servers-that-no-longer-existed-3eoh

25. **and https://en.wikipedia.org/wiki/High_availability_software**
background definitions
https://en.wikipedia.org/wiki/High_availability