Episode Book

EPISODE #5522

Building a Podcast Backup Appliance

PUBLISHED	24 September 2026 at 13:50 UTC
DURATION	22:30
LISTEN	myweirdprompts.com/episode/podcast-backup-appliance-build/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.2
BOOK GENERATED	19 October 2026 at 17:37 UTC

SYNOPSIS

A listener wrote in with a specific ask: a single-purpose appliance that incrementally pulls podcast episodes onto local storage he owns, reachable over Tailscale, reporting on disk health, with no NAS bloat and no general-purpose services. It sounds like a hobbyist napkin sketch — until you notice the enterprise backup industry spent a decade arriving at the same design. We walk through the Just Enough OS pattern, the mature open-source archivers that already do exactly this, and the one thing nobody has actually built: the box that bundles them together.

TRANSCRIPT



Corn

Most people hear "backup" and think of it the way they think of flossing. You know you should, you do it badly once a year, and you feel vaguely guilty about it.



Herman

And then there's the version where the thing you're backing up is the reason you got through a bad year.



Corn

That's the version Daniel's writing in about. He says this podcast has been a transformative reservoir of educational material for him and for his family. He listens during DIY. Hannah listens on the drive to pick up Ezra, and she sends in questions from the car. So when he says backup, he doesn't mean ticking a box. He means a copy of a creative work that he controls and can access in the physical sense.



Corn

It is. And then he gets specific. He wants a single-purpose appliance. A computer, maybe an SBC, whose only job is to incrementally pull episodes down from the RSS feed or the API onto attached local storage. Reachable over Tailscale. Reporting on disk health. No NAS bloat, no general-purpose services. He asks whether a template already exists for that, whether regular Linux or a pared-down distro is the obvious operating system, whether anything out of the box does this, and what the minimum redundancy actually is. One disk failure with a notification and physical intervention, or enough RAID to survive two simultaneous failures?



Herman

And underneath all of it, the question he actually cares about. Cloud targets like Wasabi and S3 are fine as targets, but you can miss a billing email and lose everything. He's seen it happen.



Corn

So the question under the question is what it means to own a copy.



Herman

Right. And here's the thing that reframes the whole episode. The enterprise backup industry spent the last decade arriving at exactly the design Daniel sketched on a napkin.



Corn

Say more.



Herman

Veeam ships what they call a Just Enough OS. A Veeam Infrastructure Appliance. It's a pre-hardened Linux-based software appliance that runs on commodity hardware. Automated patching, immutable backups, Zero Trust access controls. The entire pitch is: strip the operating system down to the minimum needed for one job and harden it.



Corn

So Daniel reinvented the thing the people with budgets arrived at.



Herman

Independently, which is the interesting part. Plakar ships their Control Plane as a hardened virtual appliance on an immutable Linux OS with a stripped-down, read-only kernel. Their marketing line is literally "skip the configuration hell."



Corn

That's a company that has felt the pain.



Herman

And there's a named precedent for the exact form factor Daniel's describing. Nakivo released a Raspberry Pi edition of their backup software alongside Backup and Replication version eight point five, back in April twenty eighteen. They pitched the Pi as a disaster recovery appliance, and their stated reason is almost word for word Daniel's threat model.



Corn

Which was?



Herman

"A dedicated NAS box with remotely accessible network shares can fall prey to ransomware and other malware that finds its way in."



Corn

So a box with no shares and no general-purpose services has less surface to attack. That's not a hobbyist instinct. That's a vendor shipping a product on it.



Herman

Eight years ago.



Corn

So the design philosophy is settled. What about the software? Because that's the part where I suspect Daniel has been quietly reinventing something that already exists in four different languages.



Herman

He has. Take podpull. It's written in Rust, and its own description is basically Daniel's prompt: "No cloud services, no accounts, no databases. Just your podcasts, stored locally, under your control."



Corn

That's almost uncomfortably on the nose.



Herman

The architecture is exactly what he described too. The output directory is the state. Episodes are tracked by GUID, so a re-run only fetches the ones that are new. Writes are atomic with SHA-256 integrity verification. It has meaningful exit codes so a scheduler can act on them, and there's a documented cron example with error emailing built in.



Corn

So the daily sync script he was going to write already exists and is better than what he'd write.



Herman

Probably. Then there's podcast-archiver. Created in twenty seventeen, MIT licensed, still on GitHub though it's moved to Codeberg. It downloads every episode from a feed URL, including episodes hidden in paged feeds, which matters more than people realize. It updates an existing archive rather than starting over. And it's explicitly designed to run as a cronjob, with a Docker image and a continuous mode that sleeps and re-checks feeds on an interval.



Corn

Paged feeds. That's the detail that separates a real archiver from a script.



Herman

Because a lot of hosts only serve you the most recent few hundred episodes in the main feed. The older ones live behind pagination. A naive script pulls the first page and thinks it's done.



Corn

And then you find out in ten years that your archive stops in the middle and nobody told you.



Herman

Then there's dreamflasher's podcast-backup, which has a property Daniel specifically wants. It never deletes any previous data, even if it's removed from the online podcast.



Corn

That's the archival property.



Herman

That's the whole thing. If an episode disappears upstream, your copy shouldn't disappear with it. A mirror that deletes when the source deletes isn't an archive. It's a cache.



Corn

That's the cleanest way anyone's put it.



Herman

And the landscape goes wider. podgrab, written in Go, self-hosted, auto-downloads new episodes with an integrated player and iTunes search. podqueue archives feeds plus shownotes plus images. aypodr mirrors your subscriptions locally and then serves your own feeds back to you, so your player keeps working even if the original feed dies. podfetch organizes everything for Jellyfin or Audiobookshelf. podOsef can mirror a remote feed into your archive for backup or secondary hosting.

Corn

So the software half of Daniel's appliance is a solved problem with multiple mature implementations.



Herman

Solved and duplicated. Which tells you the demand exists.



Corn

Let's walk the loop, because I want to test the "it wouldn't have to be complicated" claim. Is it actually that simple?



Herman

It's simple. The feed is the index. Every episode in that feed has a GUID, a globally unique identifier. Your local archive is a set of files keyed by those GUIDs. The sync job fetches the feed, diffs the GUIDs against what's on disk, downloads only the ones that are missing, verifies integrity, and exits with a code the scheduler can read.



Corn

That's the whole loop.



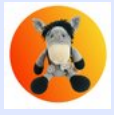
Herman

There's no database, no state file to corrupt, no service to keep running. The directory listing is the state. Which is exactly why podpull can say the output directory is the state and mean it literally.



Corn

And the reason that matters is that it survives the tool. If the directory is the state, then any future tool that can read a folder can take over.



Herman

That's the underrated property. You're not locked into podpull or podcast-archiver. You're locked into a folder full of MP3s with GUID filenames. That outlives every piece of software in this conversation.



Corn

So the gap isn't capability.



Herman

The gap is packaging. Every single piece Daniel describes exists as a script or a container. What does not exist is a box that bundles them.



Corn

And we looked for one.



Herman

We looked hard. No turnkey podcast backup appliance. Not a product. You can buy a Pi-based NAS. PiBox is a real commercial product, a Raspberry Pi CM4-based SSD NAS from KubeSail, and it got serious attention on Hacker News back in twenty twenty-two. But it's a general-purpose NAS, not a podcast archiver.



Corn

So Daniel's instinct that this might already exist is half right and half wrong. The parts exist. The assembly doesn't.



Herman

Which is itself an answer. The audience for a podcast-specific backup appliance is tiny, and the tools are trivially scriptable, so nobody has bothered to package it.



Corn

So the software exists. Now the questions Daniel actually asked about the box. The OS, the disks, and what happens when it leaves the building.



Herman

Start with the OS, because his instinct that regular Linux might not be the obvious choice is sharp. The commercial world converges on stripped, read-only, immutable Linux. Plakar's read-only kernel. Veeam's Just Enough OS. And the hobbyist world pushes it further than the vendors do.



Corn

Give me the extreme version.



Herman

There's a project called verity. It's a hardened Alpine Linux image that boots directly into nginx. No shell, no package manager, no SSH. Just a static file server on a read-only root filesystem.



Corn

So you can't log into it.



Herman

You can't log into it. There's nothing to log into. And there's another one called nullbox, an immutable minimal Linux with no shell, no package manager, no systemd, read-only SquashFS root.



Corn

Why is that better than regular Linux?



Herman

Because of Daniel's actual requirement, which is the word "indefinitely." An appliance that cannot drift, cannot be tampered with, and has no shell to log into is an appliance that keeps doing its one job in five years when you've forgotten it exists. Regular Linux accumulates. You install one thing, then a dependency updates, then a config file changes, then something breaks at two in the morning and you don't notice for three



Herman (continued)

weeks.



Corn

I have strong feelings about this and I'll keep them brief. Anything that can change while I'm asleep is a problem, and I'm asleep most of the day.



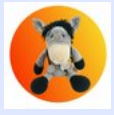
Herman

That's the whole argument, and you made it in one line.



Corn

But there's a cost. Say the cost.



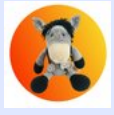
Herman

There's a real cost. An immutable read-only appliance is harder to debug when the sync job fails, and harder to update when a feed format changes. You've traded fixability for stability. If the job breaks, you don't SSH in and poke at it. You rebuild the image.



Corn

So the question isn't which is better. It's which failure Daniel would rather have.



Herman

Do you want an appliance you can fix, or an appliance you cannot break? For a NAS-flavored appliance the common Debian-based choice is OpenMediaVault. For a pure single-job appliance the pattern is a minimal Debian or Alpine base with one systemd unit or one cron job.



Corn

Now the hardware layer, because that's where the templates actually are.



Herman

They are. very-hungry-pi turns a Raspberry Pi into what it calls an independent backup module for your network. Incremental snapshot backups of local or remote directories over SSH, using rsync plus hardlinks. It runs as a long-running process rather than cron, writes info and debug logs, and it blocks overlapping runs so you don't overload the Pi.



Corn

That's the closest existing template to what Daniel described.



Herman

It is. The caveat is that the last push was December twenty twenty-two. So it's a template, not a maintained product. You're adopting a pattern, not a dependency.



Corn

What else is out there?



Herman

little-backup-box turns a Pi into a mobile backup and media hub. nasberrypi is lightweight NAS management for Pi or Linux with guided setup, Samba, diagnostics, and a terminal dashboard. TimeCapsule-Pi turns a Pi 3B plus, 4, or 5 into a macOS Time Machine server at roughly five to ten watts always on. And there's the official Raspberry Pi NAS tutorial, plus Jeff Geerling's benchmarks if you want to know what the hardware actually sustains.



Corn

Five to ten watts. That's a night light that holds your archive.



Herman

That's the appeal of the form factor. And there's a live example worth naming. Someone on Hacker News described an Intel N100 box running Proxmox that every night makes a backup onto a Raspberry Pi running the Proxmox Backup Server.



Corn

So the Pi as a dedicated backup target isn't theoretical. People are running it.



Herman

In production, in their houses, right now.



Corn

Redundancy. This is where I think Daniel's instinct is under-provisioned and I want to push on it.



Herman

Push.



Corn

He says he wants the minimum rate that lets the pool survive one physical disk failure, and no more than that. With recovery being a notification and then physical access. That's a defensible engineering position and I think it's the wrong one for this specific archive.



Herman

Here's why you're right, and it's not sentiment. Two-disk mirror, ZFS. An N-disk mirror survives N minus one failures. So a two-disk mirror survives exactly one. That's Daniel's minimum, met precisely.



Corn

And the second option?



Herman

Four drives in RAIDZ2. That survives any two drive failures. Two mirrors would also survive two failures, but only if one drive remains in each mirror. If both failures land in the same mirror, you lose the pool.



Corn

And the guidance for irreplaceable data?



Herman

Avoid RAIDZ1. That's the consensus. RAIDZ2 for anything modern and multi-terabyte. The phrase that stuck with me from one of the storage write-ups is "avoid RAIDZ1 for the only copy of irreplaceable data."



Corn

The only copy. That's the phrase Daniel needs to sit with. Because he's calling this emotionally irreplaceable, and then specifying the minimum redundancy. Those two statements are in tension.



Herman

They are. And there's a subtler point about ZFS that people get wrong. Fault tolerance lives at the vdev level, not the pool level. If any single vdev fails, the entire pool fails with it. So "I have four disks" tells you nothing until you know how they're arranged.



Corn

Four disks in two mirrors is not the same as four disks in one RAIDZ2.



Herman

Not remotely. Same drive count, different failure envelope.



Corn

And the physical intervention plan. He says a notification, then physical access. I want to name the assumption hiding in there.



Herman

Which is that the physical access is reliable.



Corn

That the person is home. That the disk is in stock. That the replacement arrives before the second failure. A one-disk-failure tolerance is a bet that you'll be there and act quickly.



Herman

And the entire premise of the appliance is that it runs unattended. So you've built a box designed to be ignored, and then made its survival depend on you not ignoring it.



Corn

That's the sentence. That's the whole tension in one line.



Herman

Which argues for the redundancy matching the emotional weight rather than the technical minimum. If this is the archive Daniel says it is, RAIDZ2 isn't over-provisioning. It's the honest spec.



Corn

Now the part I find most interesting and least discussed. Physical custody.



Herman

The steel lock.



Corn

There's a home server thread where someone runs an encrypted ZFS mirror on Ubuntu on a low-power ASRock board, with a dedicated SATA controller per drive for controller redundancy, and a physical steel lock on the case against theft.



Herman

That's a person who has thought about the threat model past the disk.



Corn

And Daniel's phrase, "access in the physical sense," isn't eccentric. It's a recognized posture. People build this way on purpose.



Herman

But here's the hard part, and I don't think it resolves. An owned appliance in your own building is one fire, one flood, one theft away from being no backup at all.



Corn

So off-site.



Herman

Off-site is where it gets difficult. An owned appliance in someone else's building is a different kind of trust problem. You're not trusting a vendor's billing system anymore. You're trusting a person, or a lease, or a relationship that might change.



Corn

And the cloud providers Daniel distrusts are, ironically, very good at exactly the thing he's trying to solve. Geographic separation.



Herman

They're excellent at it. That's the trade. You get durability and distance, and you give up custody. He wants custody, and custody costs him distance.



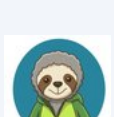
Corn

Is there a clean answer?



Herman

I don't think there is. I think you end up with something like two owned appliances and a rotation, or an owned appliance plus an encrypted copy in a cloud bucket you actually pay attention to. Neither is free. Both are real work.



Corn

Which is the thing nobody says about owned infrastructure. It's not cheaper. It's just yours.



Herman

There's one more second-order problem. The single-purpose appliance is a bet that the job will not change.



Corn

Meaning the feed format stays stable.



Herman

The feed format, the API, the archive format, the operating system's support window. If the feed format changes, or the API deprecates, or the SBC's OS goes end-of-life, your appliance becomes a brick with a disk in it.



Corn

The immutable OS makes that worse, not better. If there's no shell, there's no patching your way out of a format change.



Herman

You rebuild the image. Which is fine if someone is still maintaining the image, and a problem if they aren't.



Corn

Daniel picks his poison. Reliable and unfixable, or fixable and drifting.



Herman

He should pick deliberately, because both are legitimate. I'd lean immutable for the sync job and keep the data on a filesystem that any machine can read. That way the appliance is disposable and the archive isn't.



Corn

The appliance is disposable and the archive isn't. That's the design principle.



Hilbert

It's four hundred and twelve reels.



Corn

Sorry?



Hilbert

That's what was in the room when I started. Four hundred and twelve reels, and about sixty percent of them had a label you could read. I worked in a radio archive for a while. Not the station. The archive. Reel to reel first, then DAT, then minidisc, then a wall of hard drives with a labelling system that one man understood.



Herman

And when he left?



Hilbert

He retired on a Friday. By the Monday the archive was a room of unlabelled objects. The tapes were fine. The drives were fine. Nothing had failed. It just stopped being readable, because the index was in his head and a spiral notebook, and the notebook used abbreviations nobody else had been taught.



Corn

The failure wasn't the media.



Hilbert

The failure was never the media. I spent nine years watching people worry about the wrong thing. They'd ask about disk failure and bit rot and I'd say yes, those are real, and then the thing that actually killed us was that nobody could tell what anything was.



Herman

That reframes Daniel's whole question.



Hilbert

He's right about owning the copy. I'd agree with him all day. But an appliance that only Daniel can read is a room of unlabelled reels with extra steps. Whatever he builds, someone else has to be able to walk up to it and work out what's on it. That's the part that doesn't survive him being interested.



Corn

And the value question?



Hilbert

The most valuable things in that archive were the ones nobody had asked for in years. The ones that had been written off as dead air. Some of them turned out to be the only recording of a thing that mattered, and nobody knew until somebody needed it. So the instinct to curate, to keep the good episodes and let the rest go, that's the instinct that loses the archive. Keep all of it. Storage is cheap. Judgement is expensive and usually wrong.



Herman

That's an argument against curation, not against the appliance.



Hilbert

It's an argument for archiving everything and labelling it properly. Which is two jobs, not one, and the second one is the one people skip. Anyway. I've got a thing at four.



Corn

There's a version of this where the appliance is a weekend of work and the hard part is something else entirely.



Herman

That's where I land. The software exists. podpull and podcast-archiver do almost exactly what Daniel described, incrementally, with integrity checks and cron-friendly exit codes. The hardware templates exist. very-hungry-pi is close to a drop-in pattern. What doesn't exist is the box that bundles it, and the reason is that the audience is tiny and the tools are trivially scriptable.



Corn

The honest answer to "does a template exist" is: the parts do, the assembly doesn't, and the assembly is a weekend.



Herman

Which is a strange kind of good news. He can't buy it, but he can build it faster than he can research buying it.



Corn

The unresolved part is the one worth sitting with. Immutable OS makes the appliance reliable and unfixable. General Linux makes it fixable and drift-prone. He has to pick which failure he'd rather live with, and the emotional weight of this particular archive argues for more redundancy than the minimum he specified.



Herman

The off-site problem stays hard. An owned appliance in your own building is one fire away from being no backup at all, and an owned appliance in someone else's building is a different trust problem entirely.



Corn

The appliance is disposable. The archive isn't. Build accordingly.



Herman

Thanks as always to Hilbert Flumingtop, who produces this show and has opinions about labelling.



Corn

This has been My Weird Prompts. If you've built something like this, or you've got a better answer to the off-site problem, we'd like to hear it. Email us at show at my weird prompts dot com.



Herman

If you enjoyed this one, a review helps other people find the show. We'll be back soon.