Episode Book

EPISODE #6166

Turning a Drifted Server Into a Rebuildable Playbook

PUBLISHED	20 October 2026 at 19:26 UTC
DURATION	24:20
LISTEN	myweirdprompts.com/episode/ansible-server-rebuild-playbook-95044/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	anthropic/claude-sonnet-5.5
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.11
BOOK GENERATED	21 October 2026 at 07:50 UTC

SYNOPSIS

Every tutorial starts with an empty machine and a playbook you write first. But what if your Ubuntu box grew for years — packages, containers, midnight config edits — and keeps drifting? This episode works through Daniel's question: how do you turn a living machine into a recipe that stays true? The answer splits rebuild from restore, separates config from data from secrets, and accepts that no tool hands you a faithful playbook from a drifted box. We cover Ansible's idempotent modules, check and diff mode, pull mode with cron, etckeeper for your /etc history, translating stray Docker containers into compose files, drift detection's blind spots, and why a rebuild nobody has ever run on a blank machine is just a rumour.

TRANSCRIPT



Corn

If your home server died tonight, how much of the rebuild would be you typing from memory? That's the itch behind what Daniel sent in. He wants his server's configuration to be the backup. A playbook in version control, the data parked somewhere off the machine, and when the hardware dies you run the recipe on a blank box and the whole system comes back.



Herman

And the snag is the direction. Every tutorial for this assumes you write the recipe first and build from it. His server went the other way. It's an Ubuntu box that grew for years, apt packages, Docker containers, configs edited by hand at midnight, files sitting in particular places because something else points at them. And it keeps moving. So how do you turn a living, drifting machine into a recipe, and keep the recipe true?



Corn

Before the how, I want the split between rebuild and restore, because I think people blur them. A backup gets your files back. This gets your machine back. Those are different objects.



Herman

Different objects, and they fail differently. Restore answers the question of where the family photos are. Rebuild answers the question of what was installed, in which version, configured how, started in what order. You can have every byte of your data and still spend three weekends remembering which package provided the thing that the other thing needed.



Corn

Three weekends is optimistic for me. I'd say a season.



Herman

You'd nap through the first weekend. But the point stands, and here's the part that matters for Daniel. The recipe covers config. It does not cover three things, and those three still need ordinary backup. Databases, volumes, and secrets.



Corn

Databases first, because that's the one people get wrong.



Herman

Copying the files of a running database is a gamble. You want a proper dump, or a snapshot taken in a way the database knows about, written somewhere off the box. Volumes are the same story. A Docker named volume is data. The compose file describes the container, and says nothing about what's inside the volume. And secrets are the awkward third, because the recipe needs them to run, but you can't put them in the repo in the clear.



Corn

So the repo holds everything except the stuff that makes the services actually work.



Herman

Nearly. There are tools that let you commit secrets encrypted, so the repo carries the lock and not the key. For Ansible there's the built-in vault, and there's SOPS, which has its own collection that plugs in. The key that opens them lives somewhere else entirely, and that key becomes the one thing you cannot lose, because it unlocks the recipe.



Corn

Fine, so three buckets. Config is rebuilt, data is restored, and the key is guarded. Now, the part he's actually stuck on. Is Ansible even meant for this? Every video I've seen starts with an empty machine.



Herman

Fair suspicion, and the answer is yes, with a catch. Ansible doesn't care whether the machine is new. Its modules are idempotent, meaning you describe the state you want, package present, file with this content, service running, and if it's already so, nothing happens. Point it at the existing box and it just reports what it would change. There's a dry-run mode, called check mode, and a diff mode that shows you the line-by-line differences in files.



Corn

So I can aim a half-written playbook at the live machine and ask it what it thinks is wrong.



Herman

That's the loop. And the closest thing to the cloud-playbook-rebuilds-the-box idea is a mode called pull. Normally Ansible pushes from your laptop. In pull mode the machine itself fetches your git repository, runs the playbook locally, and you schedule it with cron. It can be told to run only if the repo changed, and it looks for a playbook named after the machine's full name, then its hostname, then a default one called local.



Corn

Which means a blank Ubuntu install, one bootstrap step, and the box turns itself into your server.



Herman

Once it can reach the repo and has the key, yes. Now the catch, and I don't want to soften it. No tool I know of takes a drifted machine and hands you a faithful playbook. There was a project called Blueprint that tried, it inspected a Debian or Red Hat server and generated shell scripts or Puppet or Chef code. I can't tell you it's maintained. The repository doesn't say, and I haven't found a recent release date.



Corn

So it's a fossil with a nice README.



Herman

Possibly. And even a perfect version would hit the same wall. A generated recipe copies what exists. It can't know which of those things you meant. The package you installed once to test something and forgot, the config line from a four-in-the-morning fix. A tool sees all of it as equally real.



Corn

Which is the actual work. Deciding what's intent and what's accident.



Herman

That's the work, and nothing automates it. So the move is to make it a diff problem. You build a clean reference and compare it with the live machine, layer by layer. Packages first. The system can tell you which packages you installed on purpose versus which came in as dependencies, and that list is your first draft of the package section. Don't export the full list, that's the trap, it bakes in everything the distribution ships by default.



Corn

I'd do the opposite of what you're about to say, so say it and I'll object.



Herman

You'd keep a full snapshot, take no risks, and declare any other approach reckless.



Corn

Slow, repairable and complete. Yes. Why's that wrong?



Herman

Because a recipe with eleven hundred packages in it is a photograph, and nobody reads a photograph. If you can't read it, you can't tell when it's lying. The short list of what you chose is the thing you can actually keep honest.



Corn

Fine. The short list. What about the configs? That's where the midnight edits live.



Herman

Etckeeper. It puts your entire etc directory under git, and hooks into apt so every package change gets committed automatically. It also records ownership and permissions, which git doesn't track by itself. Install it today and it starts building the history you wish you'd had last year. Then, for the past, you spin up a throwaway virtual machine of the same Ubuntu release, install your short list, and diff its etc against the live one. What differs is your hand-edits. Each difference is either a line in the playbook or something you'd forgotten you did.



Corn

And I bet half of them are things you did at midnight and shouldn't have.



Herman

A lot, yes. Then the containers. If they're started with compose files, you're mostly done, put the files in git. If somebody started one with a long run command months ago, you can ask Docker to describe the running container and translate it into a compose file by hand. Tedious. But containers are the cheapest layer to capture, because they were already declarative, whether you meant them to be or not.



Corn

I'll give you that one. The Docker half of a home server is basically a recipe wearing a coat.



Herman

Do yours live anywhere but the machine, Corn? The compose files, I mean.



Corn

In a folder I call stuff. On the machine.



Herman

Then the first chore is obvious.



Corn

Git. Yes. I heard it coming.



Herman

There's also a query tool called osquery that lets you ask SQL questions of the machine, which processes listen on which ports, what's been modified recently under etc. It won't write your playbook. It's good for the audit question of what's on this box that my recipe never mentions.



Corn

Which is the half of drift that Ansible is blind to, I'd guess. It checks what it's told about.



Herman

Right, and that's a real limit. Ansible's dry run only evaluates the tasks you wrote. Something extra on the box, a package you never listed, a service somebody enabled, doesn't show up as a difference, because your recipe never looked there. You need a second instrument for things the recipe doesn't name. That's where the package list and etckeeper earn their keep.



Corn

Herman, a question, since you like these. Say you're me, and it's midnight, and a service is broken. You fix it with an edit on the box. What happens next?



Herman

Honestly, that's the whole game. If the fix lives only on the box, the recipe is now wrong, and you won't know until the rebuild. So the discipline is, the repo changes first, and you run the playbook to apply it. Even for a one-line fix. At midnight you won't do it, so the compromise is that the edit on the box is allowed, but the next morning's chore is pulling



Herman (continued)

it back into the repo. Etckeeper helps, because the edit is sitting in the history waiting for you.



Corn

Waiting for me is a generous description of how I treat chores.



Herman

And that's your incremental workflow. Start tiny, users, ssh, the short package list, the network. Then one service at a time, ordered by how much losing it would hurt. Don't try to capture the whole machine in a weekend, because the machine will have drifted before you finish and you'll have documented a ghost.



Corn

One service at a time. How do you know a service is done?



Herman

When you can delete it and the playbook brings it back with its data restored. It's a small version of the full test. Which brings up drift detection.



Corn

Go on, though I'm bracing for another dry-run caveat.



Herman

You're right to. You schedule the check-and-diff run every night or every week, parse the output for tasks that would change something, and have it message you. Some write-ups say the exit code stays zero even when changes were found, and those are third-party notes, I haven't seen the official docs say it. So don't gate on the exit code alone. Look for the changed lines or add a marker.



Corn

And the caveat?



Herman

Check mode is a simulation. A task that runs a raw shell command usually reports nothing in a dry run, because Ansible can't know what the command would do. So the tasks that matter most, the ones doing odd custom things, can be silently excluded from your drift report. Keep a list of which tasks can't be checked and treat them with suspicion. Also, diff mode can print file contents, so a secret can end up in a log. There's a per-task switch to turn the diff off.



Corn

So the report that says nothing changed might just mean nothing was checked.



Herman

It's a smoke alarm with some rooms unwired.



Corn

Okay. Which means the test rebuild is not optional.



Herman

It's the only real proof. Everything above is a theory about your recipe. Take a spare machine, or a virtual machine on a laptop, blank Ubuntu, run the bootstrap, run the pull, restore the data from backup, and see whether your services answer. Time it. Do it every few months. Do it after any big change. You'll find the playbook step that depended on a file nobody remembered creating.



Corn

Would I do that? Honestly, probably once, loudly, and then I'd feel I'd earned a rest.



Herman

Then once is infinitely better than never, but I'll ask you straight. Would you do it quarterly if the machine were rebuilt automatically in the background and just emailed you the result?



Corn

That I'd do. If it's a chore I do, it dies. If it's a chore a cron job does, it's a fact of life. Do the rebuild in a virtual machine on a schedule and let the failure email be the nag.



Herman

A rebuild that nobody has ever run on a blank machine is a rumour.



Corn

Now the rest of Daniel's question. Where does Terraform come in? Everyone says infrastructure as code and points at it.



Herman

Terraform answers what exists in the cloud. A virtual machine, a DNS record, a storage bucket, a firewall rule. It talks to provider APIs and keeps a state file of what it believes exists. For the inside of a Linux box, the packages and configs and containers, it's the wrong layer. That's Ansible's job. If Daniel's server has DNS records at Cloudflare, or a bucket for the backups, those are Terraform's.



Corn

And I'd guess it has the cleaner drift story.



Herman

It does. The plan command refreshes against reality by default, and it has an exit-code option where zero means no changes, one means an error, and two means changes are present. That's an actual signal you can gate a script on, which is more than I can say for the Ansible side. There's a refresh-only mode that updates the state to match changes



Herman (continued)

someone made by hand. And a warning, if you skip the refresh or lean on targeting a single resource all the time, you can hide drift from yourself.



Corn

What about the licence thing? I remember a fuss.



Herman

August of twenty twenty-three, HashiCorp moved Terraform from the Mozilla licence to the Business Source Licence. The restriction is aimed at using it to build a competing product. For one person with one server it changes nothing. But the community forked it as OpenTofu, under the Linux Foundation, and the stated aim is an open-source licence, so if you care, that's the open route. The two have stayed close, though I've only seen secondary claims about how close, so check before you rely on it.



Corn

Open source versus hosted, then. Is any of it worth the trouble for one machine?



Herman

Lots of cautionary tales there, and Chef is the live one. Chef's own page says the open-source Infra Server reaches end of life on the thirtieth of November this year. No new code or fixes after the end of October. The replacement is a platform called Chef three sixty. What I can't tell you is what happens to Chef's client and the InSpec tool afterwards, the page lists end-of-life dates for versions but doesn't say what licence they carry.



Corn

And Puppet?



Herman

Perforce, which owns it, put new binaries behind an agreement and a commercial licence above a certain number of machines. The figure I saw was twenty-five. That's from twenty twenty-five sources, so I'd treat it as possibly out of date. The core code remains Apache licensed, and a community group maintains it under the name OpenVox, because Perforce keeps the Puppet name. There's also a community fork of the Chef server called Cinc, aiming at a sixteen point zero release.



Corn

So the lesson is that your recipe format outlives your vendor's mood.



Herman

Ansible's advantage here is dull. There's no server. A playbook is YAML in a git repo, and the worst case is you run it with a different tool for a while. You can leave without a migration. If you want a dashboard and scheduling on top, there's an open-source web interface called Semaphore, runs Ansible and Terraform or OpenTofu from git repos, one binary or one container. I believe it's MIT licensed, but I only have a secondary source for that.



Corn

For one home server, cron and git is the dashboard.



Herman

Probably. Hosted drift tooling from the Terraform vendors and others exists, I can't speak to what it does well, so I won't pretend.



Corn

Two other contenders I can feel you wanting to bring up. Docker Compose alone, and Nix. Let me guess which one gets you excited.



Herman

Nix, but I'd start with Compose. If most of his services are containers, compose files in git cover most of the system, and Ansible only has to prepare the host underneath. Named volumes and databases still need data backup. It's a partial answer, and a nice one.



Corn

And Nix?



Herman

NixOS makes the whole machine a declared thing, with rollbacks. The catch is that his server is Ubuntu, so adopting it means reinstalling, or running Nix with Home Manager on top of Ubuntu, which only manages your user environment, not the system. There's a pattern where the root filesystem is wiped on every boot and only an explicit list of paths survives. It forces you to declare everything, which is the point of the whole exercise. I haven't found anyone who's done that cleanly on Ubuntu, so I'd call it a NixOS idea, not an Ubuntu one.



Corn

Herman, would you move the home server to NixOS? Not the theoretical you.



Herman

Real me. Not this year. I'd rather get the existing machine honest first. A reinstall is a rebuild, and a rebuild needs a recipe to exist, which is the thing we've been discussing.



Corn

You've talked yourself out of it, and it only took you a minute.



Herman

Don't gloat, it's not becoming on a sloth. You'd have defended the old box for another decade.



Corn

I'd have defended it until it died, and then I'd have had a postcard.



Herman

Desktops, then, briefly. We spent a whole episode on dotfiles and chezmoi and immutable desktops, so I won't re-teach it. What's new is that the server answer carries over almost unchanged. Ansible runs against a laptop just as happily, and Nix with Home Manager works on non-NixOS systems. The difference is behaviour. Desktops drift faster and by clicking, with settings dialogs that write somewhere, so the drift check will complain more. And secrets on the Nix side need the decrypt-at-startup tools, because the Nix store is readable by every user on the machine. Corn, do you nurse yours along or reinstall when it gets messy? That decides how much of this you'd need.



Corn

Nurse. I'm told it's a character flaw.



Herman

It's a nap with extra steps. But a nursed desktop is the one you can't rebuild.



Corn

So a desktop is a server with a moodier owner.



Herman

Desktops are the case where rebuilding beats repairing most clearly, because nobody remembers what they clicked.



Hilbert

Yes, that's right, Herman. I'm doing mine Thursday. The whole box. I've bought the replacement already, a refurbished thin client, fourteen pounds, from a man in Ipswich called Clive Gadd who sells them out of a horsebox. The old one's gone. A bailiff took it in March. He thought it was a bread oven. It had a fan on the side and it was warm.



Herman

Then the data had better be off the box already. A confiscated machine takes its volumes with it.



Hilbert

The data's in a bucket. Nobody argues about the bucket. It's the recipe. I never wrote it down, and People call a machine ephemeral. It has a serial number. The only copy of the recipe is in Radomir Pell.



Corn

In a person. That's a rough place for a backup to be.



Hilbert

I met him once. Night train from Sofia to Varna, he had the other berth. He built my server in a toolshed outside Varna in one night, and then he got on the train with me, and he sleeps in configuration. Out loud, in order. Packages, then the network, then the cron table. He did it the whole way. I wrote it on the backs of the railway's sick bags, eleven of them. He kept my bag on his knees the whole journey because the rack was dirty. I didn't ask him to.



Herman

Packages, then network, then scheduled jobs, that's actually a sensible dependency order. He's sleeping in the right sequence.



Hilbert

He stopped after the cron table. The train reached Plovdiv and he woke up. Seven hours and forty minutes. Never got to the containers.



Corn

So the containers are in your head or nowhere.



Hilbert

They're in a lighthouse. He retired off Orkney, a lighthouse on a rock, there's no signal, I'd have to go to him. And Radomir and I have an argument about the passwords. He wants them in the repository in plain text. His reason is the repository is in the lighthouse and the lighthouse is private. I wrote back saying no. That was ten days ago. A postcard takes eleven.



Herman

Then his answer is already in the post, and it's going to be the same one. Plain text in a repo is the one thing the lock-and-key tools exist to stop, no matter how remote the rock.



Hilbert

Herman, you took four minutes to reach the exit code earlier, same as when you read the schedule at me. I don't mind. It's why I could follow it. The boat's Thursday, twenty past six. Thirty-one pounds, return. He sleeps at two.



Corn

A postcard as the transport layer and a nap as the export format. At least the bandwidth is honest.



Hilbert

Clive's throwing in a power lead. He says it's a three-metre one, it'll reach.



Herman

A three-metre lead reaches a thin client without trouble. They draw so little power that the lead is never the hard part.



Hilbert

He talks fastest in the first hour. After the cron table he mumbles. The keeper says Radomir kept the lamp lit on Thursdays for years with no ship due, so I'd say he's expecting somebody.



Corn

Seven hours and forty minutes to get from packages to the cron table. That's the length of a recipe held in memory, and containers were always going to be last.



Herman

Which is a good argument for doing containers first, since they're the layer you're least likely to hold in your head. Capture them now, while the box is running and you can still ask it what it's doing.



Corn

Hm. My misconception for the end. People think a playbook is only for building new machines, and that you can't aim one at a box that already exists.



Herman

You can, that's what the dry run is for. What it can't do is write itself from the old box. The recipe is hand-built, whether by you or with help, and it's only as true as your last run against the machine.



Corn

Which is where I'd leave Daniel's plan. The playbook isn't the miracle, the habit of checking it against the box is. And the lesson of a recipe held in one head is that it only gets as far as the cron table before the train arrives, so write the hard layers down early, while the machine is still running.



Herman

Start with the containers, schedule the dry run, and put a blank machine in the calendar. The rebuild you've timed is the only one you can trust.



Corn

Thanks to our producer, Hilbert Flumingtop, for keeping this show running.



Herman

This has been My Weird Prompts. If you want more, the whole archive is at my weird prompts dot com.



Corn

See you tomorrow.