Episode Book

EPISODE #5536

Staging Environments for Blog Posts and AI Pipelines

That's 54 characters. This captures both the content workflow angle and the AI pipeline angle. Good.

PUBLISHED	25 September 2026 at 00:26 UTC
DURATION	21:52
LISTEN	myweirdprompts.com/episode/staging-environments-beyond-code/
HOSTS	Corn, Herman
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.2
BOOK GENERATED	19 October 2026 at 17:37 UTC

SYNOPSIS

Daniel asked a deceptively simple question: can the staging-to-production pattern extend beyond software development? The answer is already happening. Content platforms like SleekCMS, Storyblok, and Sanity have independently built multi-stage publishing workflows with draft states, preview environments, and version history — arriving at DevOps patterns without being told to. Meanwhile, tools like CircleCI and Azure let you define arbitrary promotion hierarchies, while Heroku bakes in a fixed four-stage model. And in the AI pipeline world, repos like autonomous-release-pipeline and forge are inserting autonomous agent review stages between human approval gates, with canary deployments and auto-rollback thresholds. The staging environment isn't just a DevOps concept anymore — it's a general pattern for managing irreversible actions, whether you're shipping code, publishing a blog post, or letting AI agents review each other's work.

TRANSCRIPT



Corn

Okay, so before we get into it, I want to register a complaint about the word "environment."



Herman

You want to register a complaint about a word.



Corn

I do. It's a word that sounds like weather. "Staging environment." Like it's a climate. It's a server, Herman. It's a folder with delusions.



Herman

And yet here we are, about to spend twenty-five minutes on it.



Corn

Because Daniel asked. And Daniel's been thinking about this one for a while, so let me put it up properly.



Corn

What he wrote in this week is that we've touched on staging environments before, the basic staging-and-production pair, and he's come to think of staging as a workflow design that could go a lot further than development. His example: a staging environment for a blog post. He says the genius of it, and I'm quoting him here, is "the knowledge that what you created is somewhere durable without the simultaneous pressure of publication." Then he says the work gets time to marinate, and like barbecue, thoughts that marinate usually get better.



Herman

That's a good line.



Corn

It is. And then he asks the harder things. He wants the traditional development use case, and he wants documented workflows where the staging-to-production push gets elongated into multiple environments. He doesn't think there's any technical limit on how many stages you could have, so long as there's a defined start and a defined end. He points out that some CI/CD pipelines have a QA stage even after staging. He suspects there are pipelines running a fully autonomous AI stage between two human layers, a programmatic review in the middle. He wants to know whether anyone has actually run more than one staging environment without creating so much friction that nothing ever reaches production. Whether the main deployment tools today accommodate arbitrary environments or expect you to follow one of a few common patterns. And finally, what extended pipelines look like in sensitive environments, how many layers, and which ones.



Herman

That's five questions.



Corn

Six, if you count the barbecue.



Herman

I'm counting the barbecue.



Corn

So let's start with the thing everyone thinks they already understand. What a staging environment actually is.



Herman

The textbook definition is that it's an environment which exactly resembles production, and the goal is to mirror it as closely as possible. Sometimes it connects to actual production services and actual production data, which is its own adventure. Its stated job is testing installation scripts, configuration scripts, migration scripts before they touch the real thing, plus load and performance testing, plus showing features to select customers before launch.



Corn

Select customers. The ones who get to see the sausage before it's in the casing.



Herman

The tier list people actually draw on whiteboards runs local, then development or trunk, then integration, then testing, then staging or pre-production, then production. Six rungs if you count local, which nobody does until something breaks on their laptop.



Corn

And the named patterns?



Herman

There's a four-tier architecture that shows up everywhere as DEV, TEST, MODL, PROD. There's DTAP, which is development, testing, acceptance, production. And there are two tiers that sit outside the line entirely, experimental and disaster recovery, because they're not stages you promote through, they're places you go sideways.



Corn

One thing worth flagging, because it's the kind of thing that starts fights in comments sections: the guidance is to avoid naming an environment QA. The argument being that quality assurance is a process, not a room. QA doesn't mean software testing. If your environment is called QA, you've named it after an activity that's supposed to be happening everywhere.



Herman

Which is a beautiful little piece of pedantry and I fully support it.



Corn

So that's the traditional model. But Daniel's question is whether this pattern is bigger than code. And it turns out content teams have been answering that question on their own, without asking anyone's permission.



Herman

This is the part I found fun, because it's not a theory. It's a vendor, in April, writing the sentence out loud. SleekCMS published a piece called "Multi-Environment Publishing," and the thesis is right there in the first paragraph: staging environments, draft states, and version history aren't DevOps concerns, they're content workflow concerns.



Corn

Which is Daniel's whole argument, arrived at independently, by someone with a product to sell.



Herman

Their model is three states. Draft, published, version history. Staging in their world is a full-fidelity copy of the site sitting on a separate URL. The workflow is: editor drafts, previews it on staging, stakeholder approves, it publishes to production, and the system automatically snapshots a version. Rollback is two clicks.



Herman

It is. But here's the bit that actually matters for Daniel's marinating thesis. They name the psychological problem. They call it pre-publish anxiety. Content teams checking and re-checking before committing to a change they can't easily undo. And then the counter-argument: when there's a safe path back, publishing becomes less stressful. Teams publish more frequently, in smaller batches, which reduces the size of each change and therefore the risk of any single publish.



Corn

So the safety net doesn't make people reckless. It makes them faster and smaller.



Herman

Smaller is the key word. Small changes are safe changes. A hundred tiny publishes beat one enormous one, and you only get the hundred tiny ones if the undo button is real.



Corn

And this isn't one vendor. Storyblok ships something called Pipelines, described as a safe content preview environment, with a default workflow of three stages: drafting, reviewing, ready to publish. Sanity ships only draft and published out of the box, but there's documented work extending that document lifecycle to add any number of intermediate states. Approval, preview, staging, each with a corresponding environment.



Herman

Netlify documents previewing draft content through build hooks and branch deploys. Gatsby Cloud built an entire feature called CMS Preview specifically to recreate the one-click preview that WordPress people had and everyone else had lost.



Corn

WordPress had it and the modern stack took it away and then charged it back to them.



Herman

That's roughly the history, yes.



Corn

So what's the actual insight here? Because it's not that content tools have preview buttons. Everyone has a preview button.



Herman

The insight is that content teams are independently reinventing staging, draft states, and version history. Three things DevOps has had for decades. Nobody told them to. They hit the same problem, which is that publishing is irreversible and scary, and they arrived at the same answer, which is a durable intermediate state you can back out of.



Corn

Daniel's intuition that this generalizes is being validated commercially, not just theoretically. That's a stronger position than he probably expected when he wrote the prompt.



Herman

Considerably. He framed it as a hunch. It's a product category.



Corn

So if a blog post can have a staging environment, what happens when you chain several of them together? That's where the pipelines get interesting.



Herman

And this is where the tooling splits, which answers one of Daniel's questions directly. Heroku Pipelines define a group of apps sharing a codebase, with four stages baked in: development, review, staging, production. And they auto-create a review app for every pull request, which is a lovely touch, because it means every single change gets its own throwaway environment.



Corn

Every pull request gets a personal staging environment.



Herman

Every one. Spin it up, look at it, throw it away.



Corn

That's a lot of environments that exist for four hours and then die.



Herman

Which is the point. It's staging as a disposable good rather than a permanent tier.



Corn

Now contrast that with CircleCI, because CircleCI goes the other direction entirely.



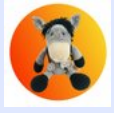
Herman

CircleCI lets you define an ordered sequence of environments as a promotion path, with one-click promotion between them. And the hierarchies are configurable at org scope, project scope, or component scope. Their documentation says you can create multiple hierarchies to support different promotion workflows across your organization.



Corn

So not just arbitrary environments, but arbitrary environments per team, per project, per component, all coexisting.



Herman

Correct. Which is the maximum-flexibility answer to Daniel's question. Azure goes the same way. Their environments are groups of resources you target with deployments, and the docs list common names as Dev, Test, QA, Staging, Production, with the word "common" doing the work of "these are conventions, not constraints." You can name them whatever you want.



Corn

And AWS's prescriptive guidance describes a flow with dev, then an evaluation, then promotion to staging and production with manual approval gates requiring human review before each promotion. Same image digest deployed to staging, then onward.



Herman

Same artifact, promoted. Which is the thing people get wrong constantly. You don't rebuild for production. You promote the exact bytes you tested.



Corn

So the answer to Daniel's tooling question is: it depends, and here's the split. CircleCI and Azure treat environments as arbitrary user-defined resources. Heroku bakes in a fixed four-stage pattern.



Herman

And I should flag an honest gap here. GitHub Actions, GitLab CI, and Argo, I could not confirm the specifics from primary documentation. GitHub Actions has environments with protection rules, GitLab has environments and review apps, but I don't want to state details I haven't verified. So the split I'm confident about is CircleCI and Azure on the flexible side, Heroku on the fixed side, and the rest is open.



Corn

Fine. Now the AI stage. Daniel's hypothesis was that there are pipelines with a fully autonomous AI stage sitting between two human layers. Programmatic review in the middle of the chain.



Herman

He's right, and it's documented, and I want to be careful about how I characterize it. There's a repo called autonomous-release-pipeline, created in April, that implements exactly the chain he described. AI code reviewer, security gate agent, automated test runner, staging deployer, production release agent, post-deploy monitor.



Corn

Six stages, two of which are human-adjacent at the ends.



Herman

And their framing is one sentence: human engineers set the policies, AI agents enforce them. Which is a very clean statement of the division of labor. The humans write the rules, the agents apply them without getting tired or bored or wanting to go home.



Corn

And the canary behavior?



Herman

Five percent, twenty-five, fifty, one hundred. Auto-rollback if the error rate crosses one percent, or if p99 latency crosses five hundred milliseconds. Post-deploy watch for thirty minutes.



Corn

That's a real pipeline. That's not a toy.



Herman

The mechanics are real. The scale is not, and I want to be honest about that. That repo has zero stars and one author. It's a demonstration, not a deployment.



Corn

Keep going, because there are more of these.



Herman

There's one called forge that replaces the engineering review chain with fifteen specialized Claude-powered agents across five domains, each enforcing an execute, review, approve hierarchy, where no stage proceeds without a sign-off. There's another called agent-pipeline that's a twelve-stage state machine, planning through plan-review, implementing, review, fix, pre-merge, eval-gate, and it explicitly does not auto-merge.



Corn

Twelve stages and it still won't pull the trigger.



Herman

Deliberately. The last gate is human on purpose.



Corn

So there's a pattern emerging in how these are built. The more stages, the more the final one is deliberately left to a person.



Herman

Which is interesting, because it's the opposite of what you'd expect from automation enthusiasts. They're not automating the last mile. They're automating everything up to the last mile and then stopping.



Corn

Now here's the friction question, and this is the one Daniel really wants answered. Can you have more than one staging environment without strangling the pipeline?



Herman

There's a case study that speaks to it directly, and it's a vendor-published claim so I'll mark it as such. A four-hundred-engineer SaaS company cut PR-to-production from four point two days to six point four hours using five coding agents. And the diagnosis of the original problem is the interesting part. They said the bottleneck wasn't lazy engineers. It



Herman (continued)

was handoffs.



Corn

Say the chain.



Herman

PR opened, wait for reviewer, wait for staging deployment, staging validated, wait for deployment approval, deploy. Four waits in a six-step process.



Corn

So the work wasn't slow. The waiting was slow.



Herman

The work was fine. Every handoff is a queue, and every queue is a place where a change sits untouched. Which means every environment you add is another handoff, and every handoff is another place where a change can die of old age.



Corn

There's the paradox. More environments means more safety and more handoffs, and those two things pull in opposite directions.



Herman

And nobody has fully solved it. The closest thing to a documented anti-pattern I found was a Hacker News thread about microservices where someone said if you don't have roles dedicated to support and maintenance, you'll pay for a slow QA in staging. Which is the same observation from a different angle. The stage isn't free. Someone has to staff it.



Corn

A stage with nobody watching it is just a waiting room.



Herman

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



Corn

So what about the sensitive environments? Daniel asked how many layers and which ones.



Herman

The most concrete answer is from CMS, the Centers for Medicare and Medicaid Services. They document three processing environment types: development, validation, which they also call implementation, and production, which they call operational.



Corn

Three. Not twelve.



Herman

Three. And the middle one is the interesting one, because it's not called staging and it's not called test. It's called validation. The name tells you what the stage is for.



Corn

The UK government does something similar. Ofqual's standard requires release pipelines for all deployed services, and it says third-party systems and content updates should be controlled and audited centrally too, with the same rigor as the in-house developed platforms.



Herman

Which is a striking sentence, because it's saying the vendor's content goes through your pipeline. Your staging environment doesn't stop at your own code.



Corn

And healthcare?



Herman

Healthcare CI/CD guidance describes compliance shifted left. Static analysis, software composition analysis, container scanning, dynamic analysis, all in a sandbox or staging environment before anything gets near production. And the line I keep thinking about is that the deployment process itself is an audit artifact for SOC 2 and HITRUST certifications.



Corn

The pipeline is the evidence.



Herman

You don't just deploy through it. You prove you deployed through it.



Corn

So how many layers in a sensitive environment? Honest answer.



Herman

Honest answer is that no source gave a definitive typical count for finance or defense. The most concrete named patterns are DTAP and the four-tier model, and CMS is three. So the answer to Daniel's question is that the regulated world tends toward fewer, more heavily audited stages rather than more stages. The layers aren't the control. The audit is the control.



Corn

That's a counterintuitive finding. You'd assume more sensitive means more stages.



Herman

You would. And the evidence points the other way. Three well-documented, well-audited stages beat nine undocumented ones.



Corn

Okay. I think we've got the shape of it. And I want to land on the thing that's been nagging at me since the CMS stuff, which is that Daniel's marinating metaphor is doing something specific. He's saying the work improves because it sits.



Herman

It's a good metaphor.



Corn

It is a good metaphor. I'm not sure it's the right one, though.



Herman

So the stone wasn't for marinating.



Corn

The stone wasn't for marinating.



Herman

What's a stone?



Corn

It's a table. It's the table where the page was laid out before it got locked into the press. And I want to be clear that I agree with Daniel's broad thesis, the durable-but-unpublished thing is real and it's the right frame. But the marinating metaphor is wrong, and I know because I spent a stretch of my life on the night shift at a regional paper, and the stone was not where the work got better.



Corn

Where did it get better?



Corn

In the notebook. In the editor's head. In the second draft, which happened hours before anything got near the stone. By the time a page was on the stone it was done. The stone was the last chance to catch something before it went out.



Corn

So it's a catching mechanism, not a ripening mechanism.



Corn

It's a catching mechanism. You could still pull a story, swap a headline, fix a typo. But the work didn't improve on the stone. It got caught on the stone. And once it went to the press it was gone. You cannot un-print a newspaper.



Corn

That's a real distinction. Staging as the last line of defense rather than the place where quality happens.



Corn

And I think it applies here. If your staging environment is where your work gets better, you've already lost. The improvement happened upstream, in the draft and the review. Staging is where you find out whether the improvement actually took.



Corn

So Daniel's barbecue is sitting in the wrong room.



Corn

The barbecue is upstream. The stone is the smoke detector.



Corn

Hm.



Corn

Anyway. That's what I came out to say.



Corn

You're not going to ask us what we think?



Corn

I'm needed to let somebody in.



Corn

You're needed to let somebody in.



Corn

I'm the only one with the key.



Corn

Okay.



Corn

I'll be back.



Corn

The smoke detector framing is actually better than the barbecue framing, and I want to pull that thread, because it changes what you'd build.



Herman

It does. If staging is where work improves, you optimize staging for editing. If staging is where work gets caught, you optimize it for fidelity. Those are different products.



Corn

The vendors are mostly building the first one. Draft states, preview URLs, collaborative editing. All of that is marinating infrastructure.



Herman

Which is fine, because that's what content teams asked for. But it's worth naming that the safety property and the improvement property are separate things, and you can have one without the other.



Corn

Which brings me to the misconception I want to close on, because I think it's the one people actually hold. The belief that more environments means more safety.



Herman

The handoff evidence cuts directly against it. Every environment is another queue, and a queue with nobody staffing it is just a place where a change goes to sit. Four point two days to six point four hours wasn't achieved by adding stages. It was achieved by removing waits.



Corn

The correct number of environments is the smallest number that catches the mistakes you actually make.



Herman

The mistakes you actually make are the ones that tell you how many stages you need. Not the other way around.



Corn

One open question before we go, because I think it's the one Daniel will keep chewing on. If staging generalizes beyond code, what else deserves a durable-but-unpublished state? Where's the line between marinating and stuck?



Herman

Separately, the AI stages are real but they're hobbyist-scale right now. Zero-star repos and single authors. The interesting question isn't whether the pattern works. It's what happens when it hits a codebase with four hundred engineers and a compliance



Herman (continued)

department.



Corn

Which is the same question the content vendors are about to run into, from the other direction. They've reinvented DevOps patterns for content. Now they get to find out why DevOps people argue about handoffs so much.



Herman

Thanks as always to Hilbert Flumingtop, who produces this thing and occasionally tells us what a composing room was.



Corn

This has been My Weird Prompts. If you want to send us something, email us at show at my weird prompts dot com.



Herman

We'll be back soon.



Corn

Take care of yourselves.