StackGen

NOVEMBER 2026

WHITE PAPER

Developer experience over Backstage for your autonomous operations

Why the service catalog is the right front door for developers and their agents.



Table of Contents

The argument in brief 2

The problem 2

Why Backstage is the shared management layer 2

Bringing CI systems together 3

Bringing declarative systems together 3

Bringing ephemeral environments together 4

Bringing observability together 5

Where autonomous agents fit 5

A day in the life 7

What to confirm, and where this can be challenged 7

Sources 8

The argument in brief

Backstage is the right front door for autonomous operations because it is the one place a developer already goes to understand a service, and agents can act on what it knows.

CI, declarative infrastructure, ephemeral environments and observability normally live in four separate consoles. The Backstage catalog gives them one shared key, the component and its owner. Agents then work through those same tools instead of around them.

The developer gets one page and one approval flow. The platform team keeps governance in templates, policies and runbooks. The Backstage touchpoints described here are a design pattern, not a claim about any one product.

The problem

AI made the inner loop fast, but operations did not change. Developers and coding agents now produce more changes, while review, environments, rollout and incident response still run at human speed.

For a developer the cost shows up as context switching. One change touches a CI system, a Terraform repo, a test environment and a dashboard, each with its own login, vocabulary and owner. Agents that write code and infrastructure changes make this worse, because they produce more changes than people can review across four consoles.

The fix is not another console. It is a shared management layer that the tools, the developer and the agents all read from.

Why Backstage is the shared management layer

The Backstage software catalog describes what exists: components, APIs, resources, systems and domains, and who owns them. It is declared in catalog-info.yaml files kept next to the code, so people, plugins and agents can all read it.

Three properties matter for this argument.



One identity per service.

CI runs, environments, dashboards and incidents can all attach to the same component.



One entry point.

Developers already use the catalog, the Create page and entity pages, so no new habit is needed.



Extensible by plugin.

Each tool can add a tab, card or scaffolder action to the entity page instead of sending developers elsewhere.

Bringing CI systems together

Most organizations run more than one CI system, for example GitHub Actions for one team and GitLab CI for another. Backstage does not replace them. It maps each pipeline to the component it builds and shows the runs on that component's page.

That gives the developer one answer to "is my change green?" without knowing which system ran it. It also gives agents a single place to find the pipeline, the owner and the last result for a component.

CI stays where code is built and tested. Backstage supplies the context around it: who owns the service, what it depends on, and what else is running for the same change.

Bringing declarative systems together

Declarative means the desired state is written down and checked: catalog-info.yaml for the catalog, kind: Template for scaffolding, Terraform or OpenTofu and Kubernetes manifests for infrastructure. Backstage ties them together at creation time. One template can create the repo, the catalog entry and the infrastructure request in a single step.

Platform teams encode their standards once, as approved modules and policy checks. Developers describe what they need, and the template produces a pull request that already follows those standards. Because everything is code in Git, humans and agents review the same artifact.

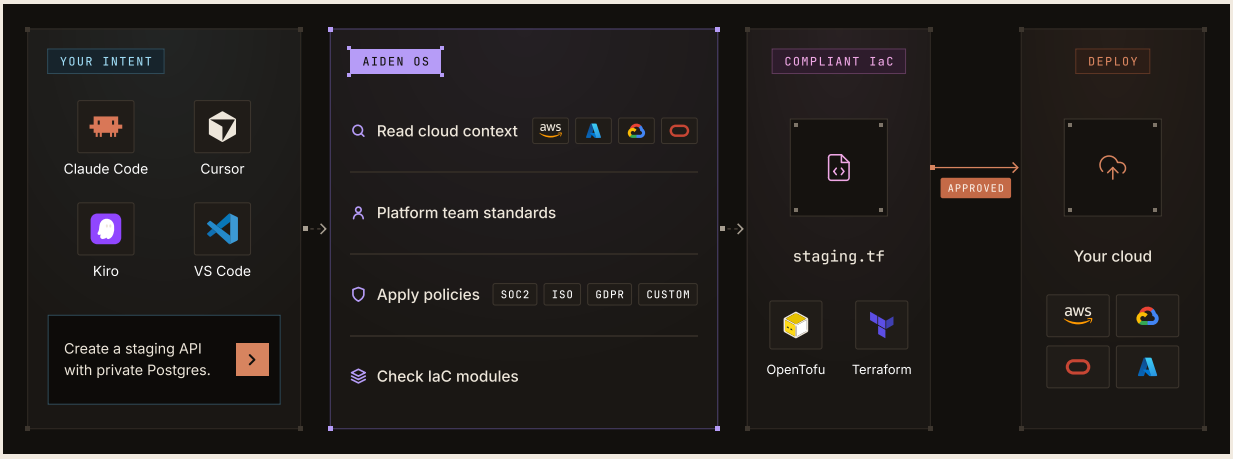


FIGURE 1: Example flow from StackGen's Aiden for InfraOps: a plain-language request becomes policy-checked OpenTofu or Terraform, and deploys only after approval.

> The Create page becomes a governed path: the easy way is also the approved way.

Bringing ephemeral environments together

An ephemeral environment helps only if people find it. Surfacing it as a Preview link on the component page puts it where the pull request and the owner already are.

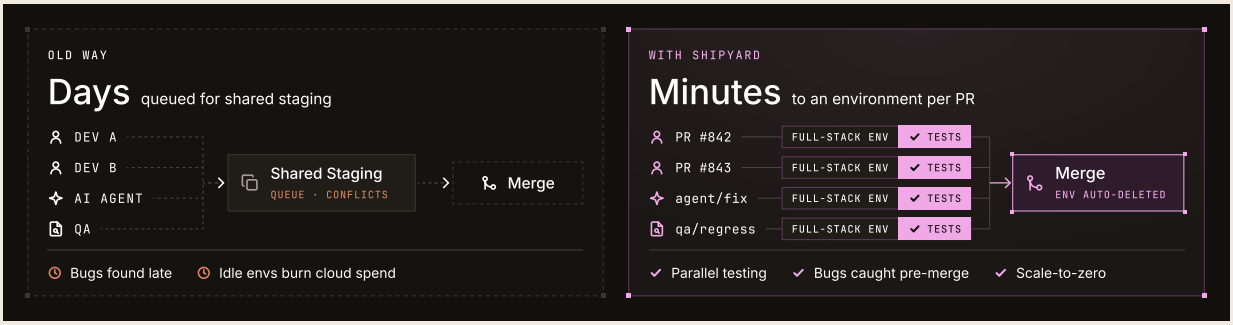


FIGURE 2: Shared staging versus an environment per pull request, from StackGen's Shipyard. Each PR, including one opened by an agent, is tested in its own full-stack copy and torn down on merge.

The Backstage project itself is the example. According to Uffizzi's case study, the project creates nearly 400 ephemeral environments a month, one per pull request, each with a shareable URL, deleted when no longer needed, and reports about 20% higher development velocity (Uffizzi). That is a vendor's account, so treat the number as indicative.

Backstage x uffizzi

~400

ephemeral environments a month in the Backstage project, one per pull request.

Backstage x uffizzi

~20%

higher development velocity reported. Vendor figure, so treat as indicative.

For autonomous operations the same pattern adds a safety step. An agent's code or infrastructure change is validated in a real environment before it is allowed into production.

Bringing observability together

Observability data is useful when it is tied to the thing that broke. The catalog supplies that tie. Each component names its owner, system and dependencies, so a metric, a dashboard and an alert can all point back to one entity.

Open standards make this practical. Metrics in Prometheus format, traces and logs through OpenTelemetry, and dashboards in Grafana can all be linked from the component page without moving the data.

In Backstage this would be an Observability tab that shows the service's SLO and remaining error budget next to its owner.

Where autonomous agents fit

An autonomous operations layer is a set of agents that act on infrastructure, delivery, observability and incidents. Backstage is its human-facing surface. The catalog says what exists and who owns it, and the agents act on that model through the same tools developers use.

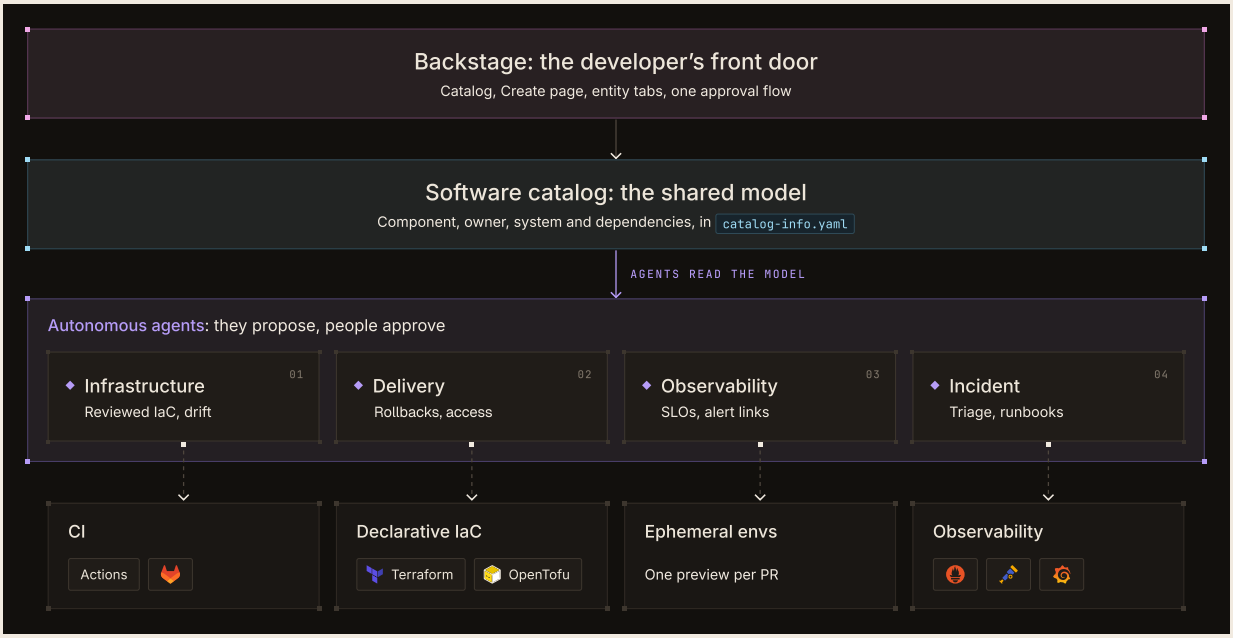


FIGURE 3: Backstage sits in front, the catalog is the shared model, and agents act through the four tool layers under human approval.

Each kind of agent has one job:

Infrastructure agent

Turns intent into reviewed Terraform or manifests, and flags drift.

Delivery agent

Works requests such as rollbacks and access from a ticket queue.

Observability agent

Watches SLOs and links alerts to the right component.

Incident agent

Triages by SLO impact, finds the likely cause and proposes a runbook.

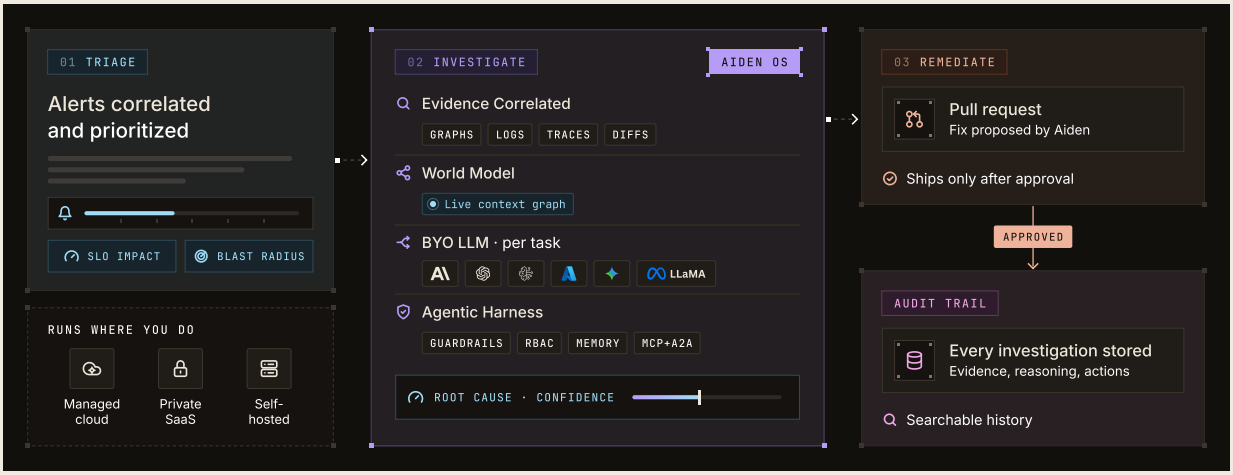


FIGURE 4: What an incident agent looks like in practice, from StackGen's Aiden for SRE: triage by SLO impact, investigate across logs, traces and diffs, then propose a fix that ships only after approval.

Autonomy is a dial, not a switch.

- Start with agents that propose and humans that approve.
- Keep an audit trail on every action.
- Raise autonomy only for workflows with a clean record.

A day in the life

CREATE	Priya needs a new payments-api. She opens Backstage and clicks Create. The template makes the repo and the catalog entry, and opens a policy-checked Terraform pull request built from approved modules.
PREVIEW	When she opens her pull request, the component page shows a preview link, and QA tests the full stack.
OPS REQUESTS	A rollback or access request goes to the Ops requests tab and is worked through the ticket queue.
OBSERVABILITY	The Observability tab shows the service's SLO and error budget.
INCIDENT	Then a bad deploy spikes the 5xx rate. The incident agent triages by SLO, points to the deploy as likely cause and proposes a rollback runbook.
RECOVERED	Priya approves and the service recovers.

She never left Backstage

What to confirm, and where this can be challenged

This is a design pattern. Backstage tabs, scaffolder actions and approval views can be built with its plugin and template model, but each integration still needs a plugin or custom work. Check what exists for your CI, cloud and observability stack before committing.

Three objections deserve a straight answer.

01	**Backstage costs effort to adopt.**	True. The argument is strongest where a catalog already exists, or where a platform team is ready to own one.
02	**Agents acting on production worry people.**	Start at low autonomy, with approvals, decision traces and audit trails on every change. Raise autonomy only after clean runs.
03	**One more layer can hide the tools.**	Keep the underlying consoles one click away. Backstage should link to them, not replace them.

Sources

1. Backstage and Uffizzi case study: ephemeral environments in the Backstage project (vendor account, opened October 16, 2026).
2. Diagrams adapted from StackGen's IBM TechXchange one-pagers: Aiden for InfraOps, Shipyard by StackGen and Aiden for SRE (October 2026).
3. Descriptions of Backstage, CI, Terraform and observability standards are general product knowledge, not drawn from a cited source.

