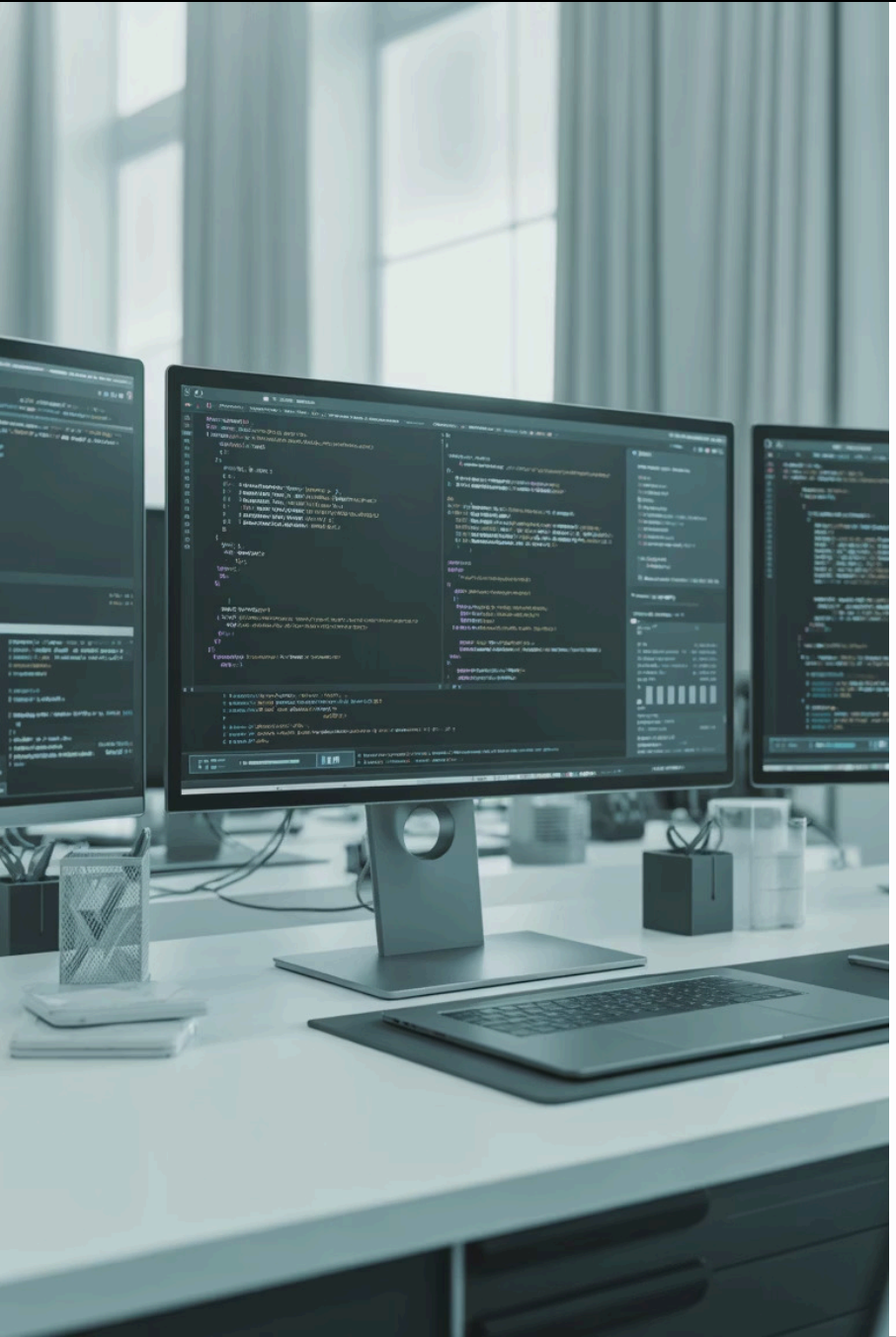


Backstage Extensions & GitHub Integration

A comprehensive guide to building powerful internal developer platforms through extensibility, automation, and seamless integrations

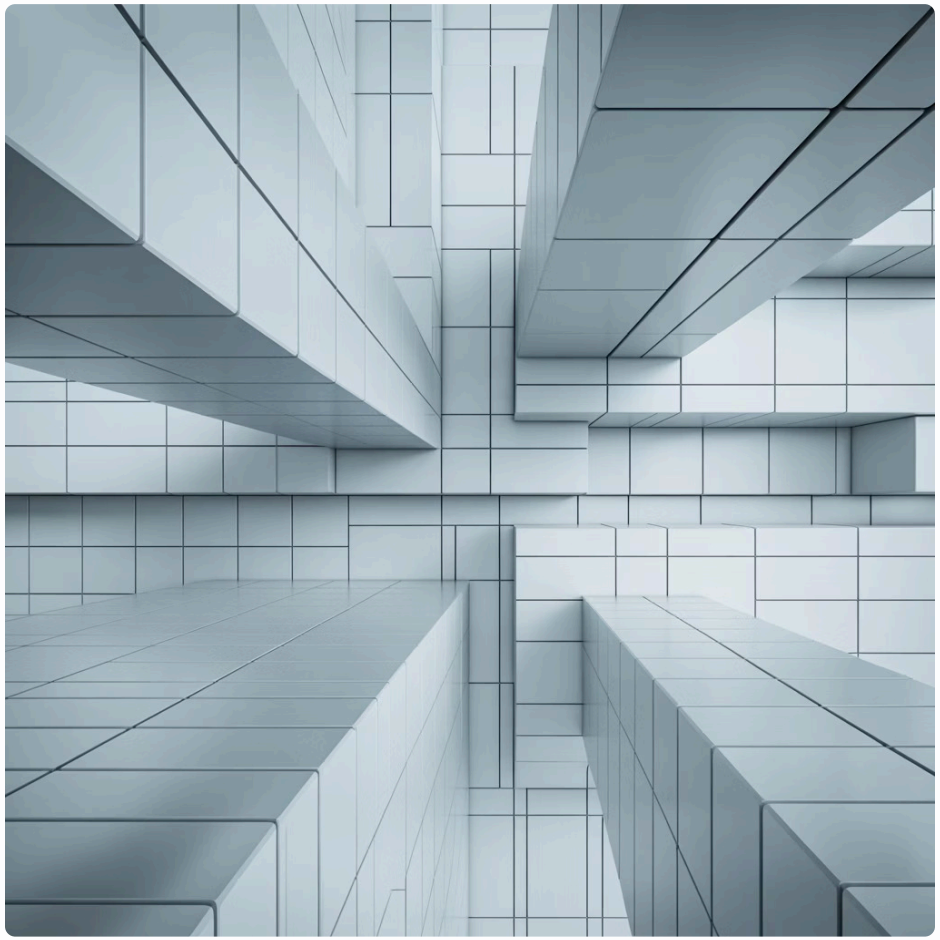
Plugin Architecture

The foundation of Backstage's power

What Makes Backstage Extensible?

Backstage is built on a **plugin architecture** that allows teams to customize and extend functionality without modifying the core platform. Think of it like building with LEGO blocks—each plugin adds specific capabilities while working seamlessly with others.

This architecture enables organizations to create internal developer portals that perfectly match their unique needs, tools, and workflows. Whether you need to integrate with Kubernetes, display CI/CD pipelines, or show security scan results, there's likely a plugin for that.



Types of Backstage Plugins

Frontend Plugins

UI components that appear in the Backstage interface

- Service catalog views
- Tech documentation pages
- Custom dashboards

Backend Plugins

Server-side functionality and API integrations

- Data fetching from external systems
- Authentication handlers
- Webhook processors

Scaffolder Actions

Custom steps for software templates

- Repository creation
- Infrastructure provisioning
- Configuration file generation

Real-World Plugin Example: Tech Radar

The Tech Radar plugin visualizes your organization's technology landscape. Imagine you're a new developer joining the team. Instead of asking "What database should I use?" or "Which frontend framework is approved?", you simply check the Tech Radar.

It displays technologies in rings (Adopt, Trial, Assess, Hold) helping teams understand which tools are recommended, which are being evaluated, and which should be avoided. This creates instant clarity and prevents teams from accidentally building with deprecated technologies.

How Plugins Communicate

Frontend Plugin

Renders UI component requesting service data

Backend API

Processes request and fetches from external systems

External Service

GitHub, Jenkins, or other integrated platform

This separation allows plugins to remain modular and maintainable. Frontend plugins focus on user experience, while backend plugins handle complex integrations and data transformations.

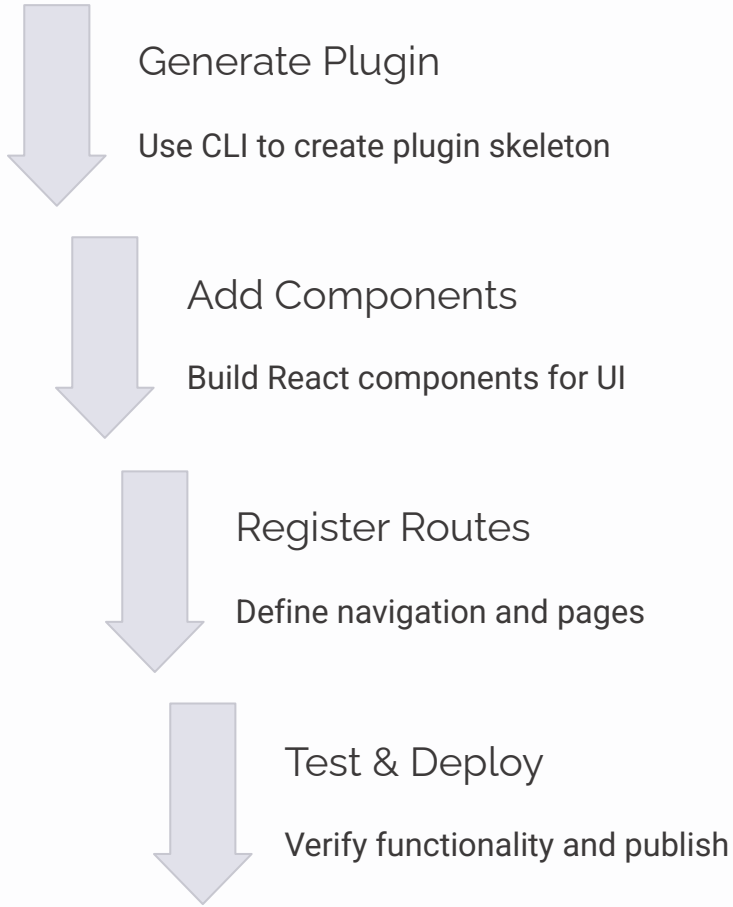
Building Your First Plugin

The Basics

Creating a plugin starts with a simple command:

```
yarn backstage-cli create-plugin
```

This scaffolds a basic plugin structure with all the necessary files and configurations. From there, you add your custom functionality.



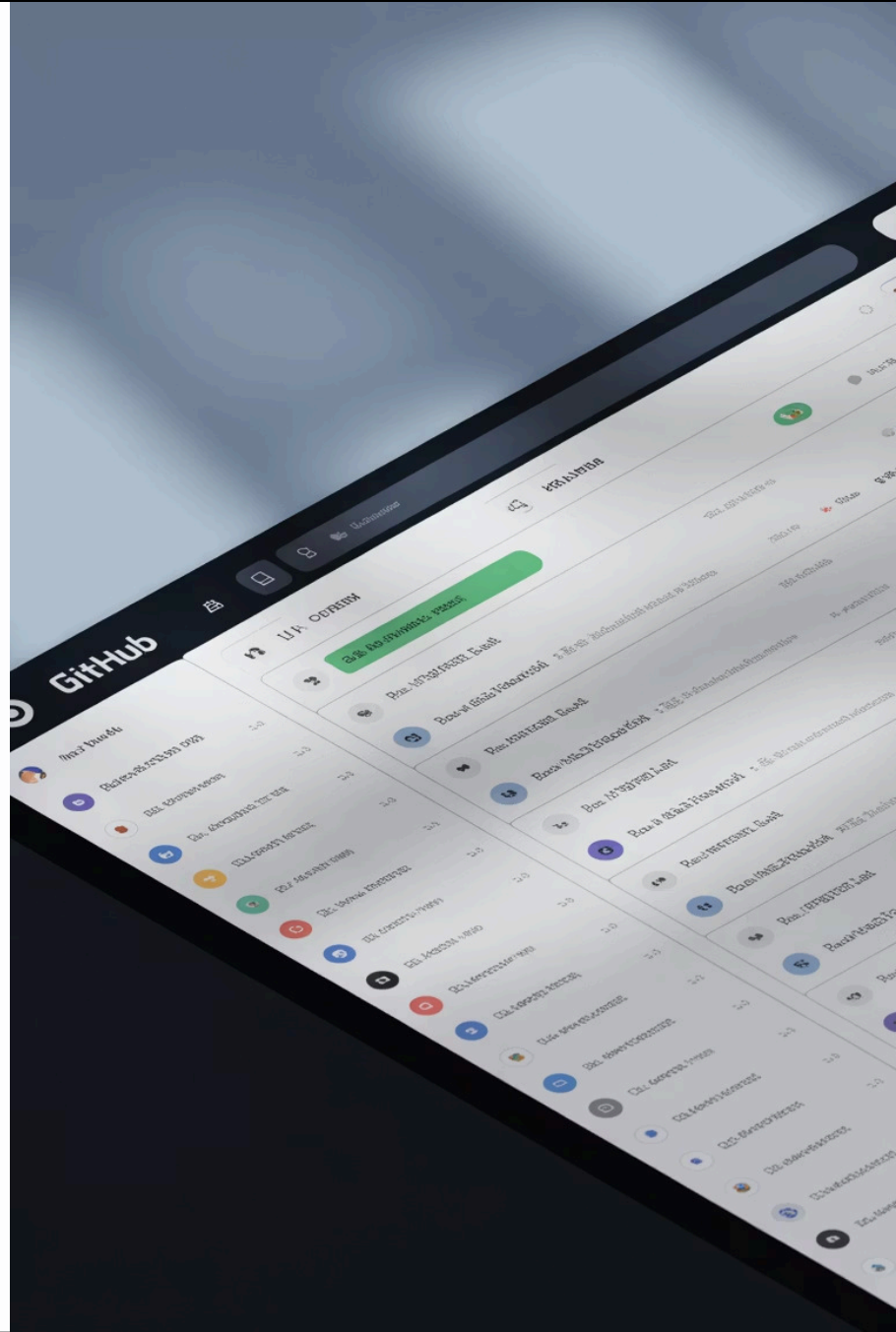
GitHub Integration

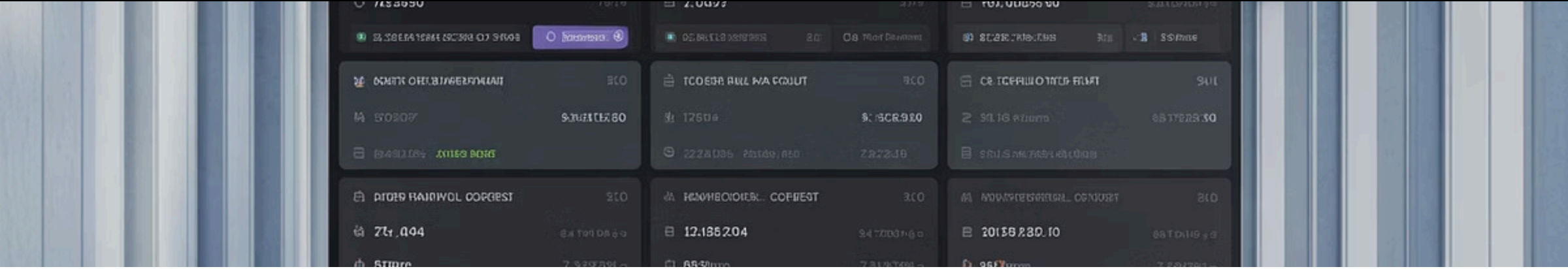
Connecting your code repositories

Why Integrate GitHub with Backstage?

GitHub integration transforms Backstage into a unified hub for all your development activities. Instead of context-switching between tools, developers can view repository health, pull requests, workflows, and security alerts directly from the service catalog.

For example, when a developer opens a service page in Backstage, they immediately see: recent commits, open PRs awaiting review, CI/CD pipeline status, and security vulnerabilities—all without leaving Backstage. This reduces cognitive load and accelerates development velocity.





Real-World Example: PR Dashboard

Imagine you're a tech lead managing five services. Instead of manually checking each repository for pending pull requests, your Backstage home page displays a consolidated PR dashboard.

You see: PRs awaiting your review highlighted in yellow, PRs blocked by failing tests in red, and PRs ready to merge in green. Clicking any PR opens it directly in GitHub. This single view saves 30 minutes daily and ensures nothing falls through the cracks.

Golden Paths

Paving the way for developer success

What Are Golden Paths?

A **Golden Path** is the opinionated, well-supported route for accomplishing common development tasks. It's the "happy path" that includes best practices, approved tools, and automated setup.

Think of it like a GPS navigation system for developers. Instead of figuring out which database to use, how to set up CI/CD, or where to deploy, the Golden Path provides a clear, tested route that just works.

For example, your Golden Path might be: "Use Node.js with TypeScript, PostgreSQL for data, GitHub Actions for CI/CD, and deploy to Kubernetes." New projects automatically get this proven stack.

Why Golden Paths Matter

Reduce Decision Fatigue

Developers don't waste hours researching which tools to use or how to configure infrastructure

Standardization

All services follow the same patterns, making them easier to understand and maintain

Faster Onboarding

New team members are productive immediately because everything works the same way

Built-in Best Practices

Security, monitoring, and compliance are included by default, not afterthoughts

Designing Effective Golden Paths

Understand Developer Needs

Survey your teams to identify common pain points and repetitive tasks

Choose Your Tech Stack

Select proven technologies that balance innovation with stability

Automate Everything

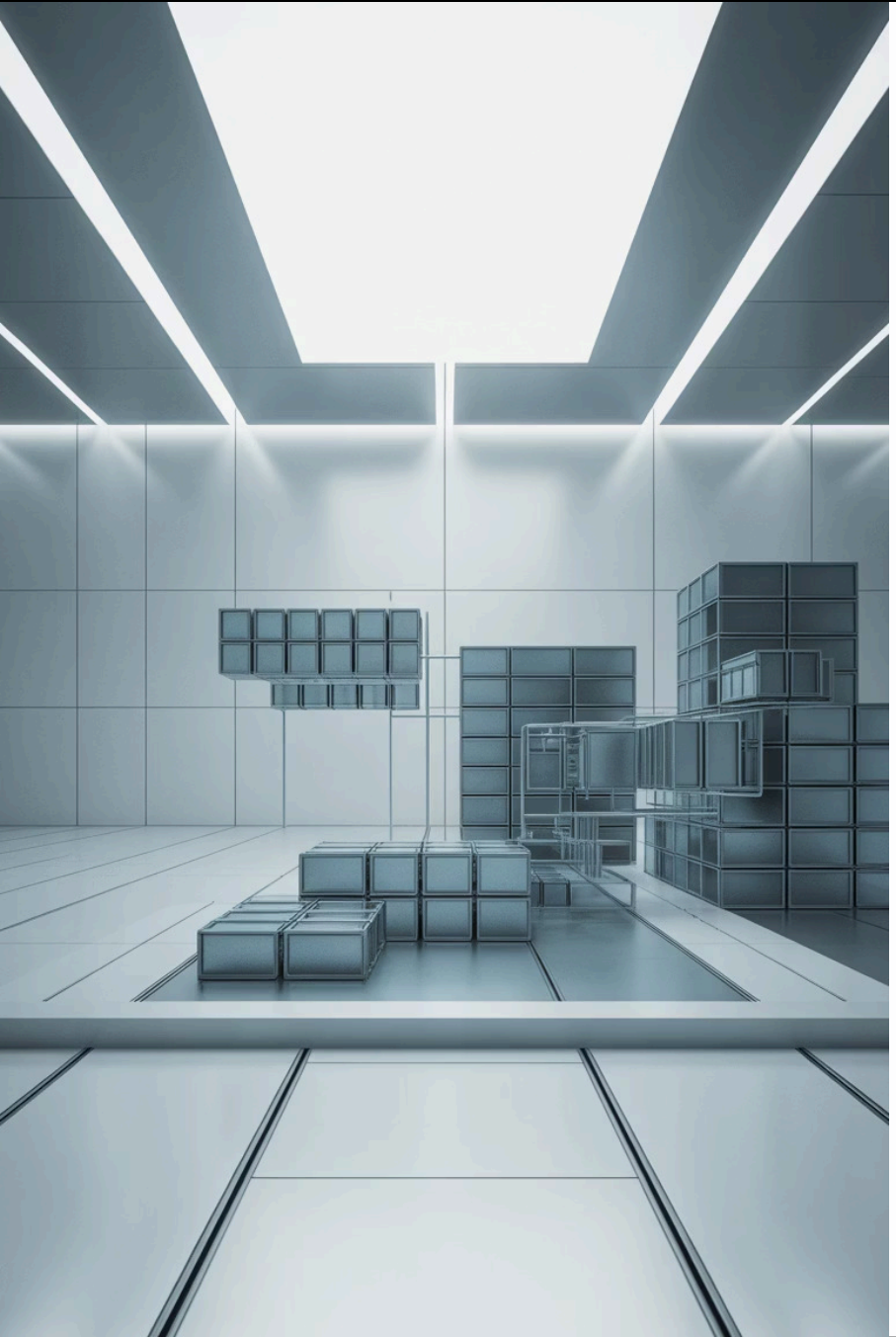
Build templates that generate fully-configured projects in minutes

Document the Path

Create clear documentation explaining why decisions were made

Iterate Based on Feedback

Continuously improve based on real-world usage and developer input



Golden Path Example: Microservice Creation

Let's say a developer needs to create a new REST API microservice. Without a Golden Path, they spend days deciding on frameworks, setting up Docker, configuring CI/CD, adding monitoring, and implementing authentication.

With a Golden Path, they run a single Backstage template, answer a few questions (service name, team ownership, database needs), and within 5 minutes have a production-ready service with: fully configured repository, CI/CD pipeline running tests, Kubernetes deployment manifests, monitoring dashboards, and authentication middleware. They can start writing business logic immediately.

Common Golden Paths to Implement

1

New Service Creation

Scaffold microservices, APIs, or web applications with all infrastructure code

2

Library or Package

Generate reusable code libraries with publishing pipelines to internal registries

3

Documentation Site

Create technical documentation sites using approved tools like MkDocs or Docusaurus

4

Infrastructure Resources

Provision databases, queues, or storage using Infrastructure-as-Code templates

Developer Self-Service

Empowering teams to move independently

The Self-Service Philosophy

Developer self-service means empowering engineers to provision resources, deploy services, and solve problems without waiting for other teams. It's about removing friction and reducing dependencies.

Instead of submitting a ticket to DevOps and waiting three days for a database, developers click a button in Backstage, answer a few questions, and have their database provisioned in minutes—with backup policies, monitoring, and security already configured.

This doesn't mean chaos. Self-service works within guardrails: developers get freedom within approved patterns, security policies are enforced automatically, and everything remains auditable.

Service Scaffolding with Software Templates

Backstage Software Templates are the engine of self-service. They're interactive forms that generate complete, production-ready projects based on your organization's standards.

A template might ask: "What's your service name?", "Which team owns this?", "Do you need a database?", and "Which API style: REST or GraphQL?" Based on these answers, it generates a fully configured repository with appropriate code, tests, CI/CD pipelines, and deployment configurations—all following your Golden Path.

Anatomy of a Software Template

Input Parameters

Form fields collecting information from the developer

Template Actions

Steps that execute: fetch templates, replace variables, create repos

Output Resources

Generated repository, CI/CD pipelines, service catalog entries

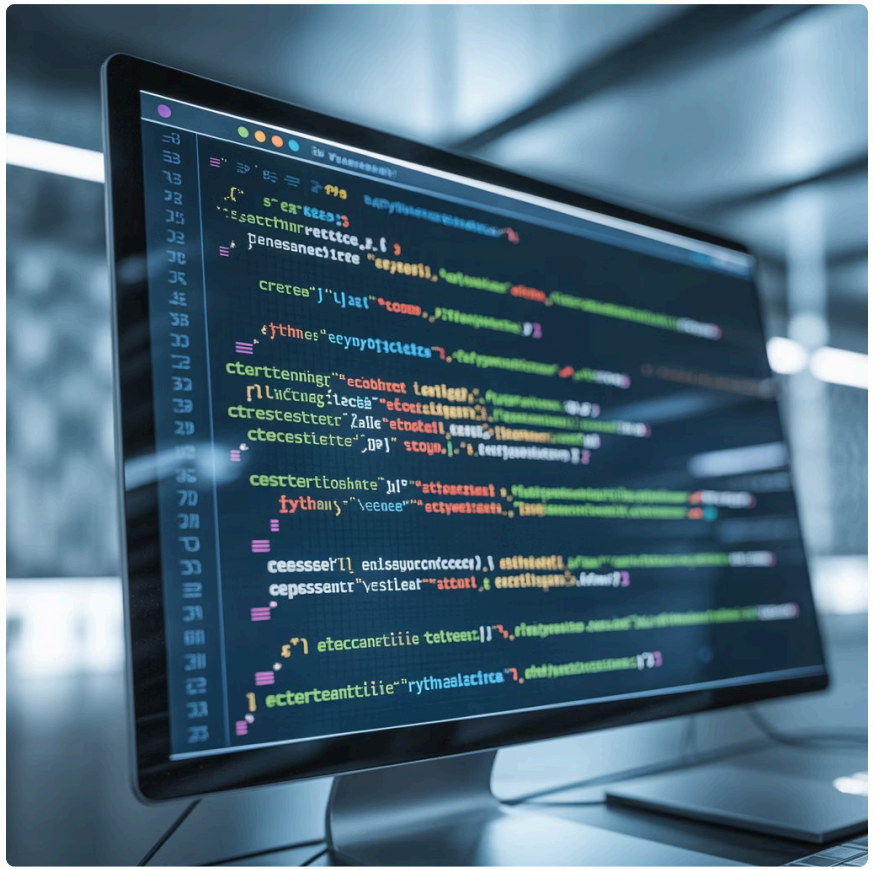
Each template is defined in YAML and can include custom actions specific to your infrastructure and tooling.

Real-World Template Example

Python API Service Template

This template creates a FastAPI service with:

- Poetry for dependency management
- Pytest test suite with 80% coverage
- GitHub Actions CI/CD pipeline
- Dockerfile for containerization
- Kubernetes deployment manifests
- Prometheus metrics endpoints
- OpenAPI documentation



A developer fills out a 5-field form and gets a complete service they can deploy immediately. First commit to production in under 30 minutes.

The Backstage Plugin Ecosystem

Backstage has a thriving ecosystem of over 100 open-source plugins that extend its capabilities. These plugins integrate with popular tools and services across the development lifecycle.

Kubernetes Plugin

View pod status, logs, and resource usage for your services directly in Backstage

SonarQube Plugin

Display code quality metrics, vulnerabilities, and technical debt analysis

CI/CD Plugins

Integrate Jenkins, CircleCI, GitHub Actions, or GitLab pipelines

PagerDuty Plugin

See on-call schedules, incidents, and escalation policies for services

Grafana Plugin

Embed monitoring dashboards and metrics charts within service pages

Lighthouse Plugin

Track website performance, accessibility, and SEO scores over time

Building vs. Adopting Plugins

Use Existing Plugins

When: Community plugins meet your needs with minimal customization

Benefits: Faster implementation, community support, regular updates

Example: GitHub, Kubernetes, ArgoCD plugins work out-of-the-box

Build Custom Plugins

When: Integrating proprietary internal tools or unique workflows

Benefits: Perfect fit for your needs, competitive advantage

Example: Custom deployment approval workflows, internal cost tracking

Measuring Success: Key Metrics

75%

Reduction in Time-to-Deploy

New services go from idea to production in hours instead of weeks

90%

Developer Satisfaction

Teams report higher happiness due to reduced friction and context-switching

5min

Average Template Usage

Developers create new services in under 5 minutes using Golden Paths

50%

Support Ticket Reduction

Fewer requests to DevOps teams as developers self-serve infrastructure

Getting Started: Your Action Plan

1

Deploy Backstage

Set up a basic Backstage instance and populate your service catalog

2

Integrate GitHub

Configure OAuth and enable GitHub plugins for repository visibility

3

Create First Template

Build a simple software template for your most common service type

4

Add Essential Plugins

Install plugins for your CI/CD, monitoring, and cloud infrastructure tools

5

Define Golden Paths

Document and automate your approved technology stacks and patterns

6

Gather Feedback

Release to a pilot team, iterate based on real usage, then expand

Key Takeaways

Extensibility is Power

Backstage's plugin architecture allows you to create a tailored developer portal that integrates all your tools in one place

Golden Paths Accelerate

Well-designed Golden Paths eliminate decision fatigue and let developers focus on building features instead of configuring infrastructure

Self-Service Scales

Software Templates and self-service tools empower developers to move independently while maintaining standards and security

Start small, iterate quickly, and continuously gather feedback from your development teams. The goal is to make developers' lives easier—if they're not using it, ask why and adjust.