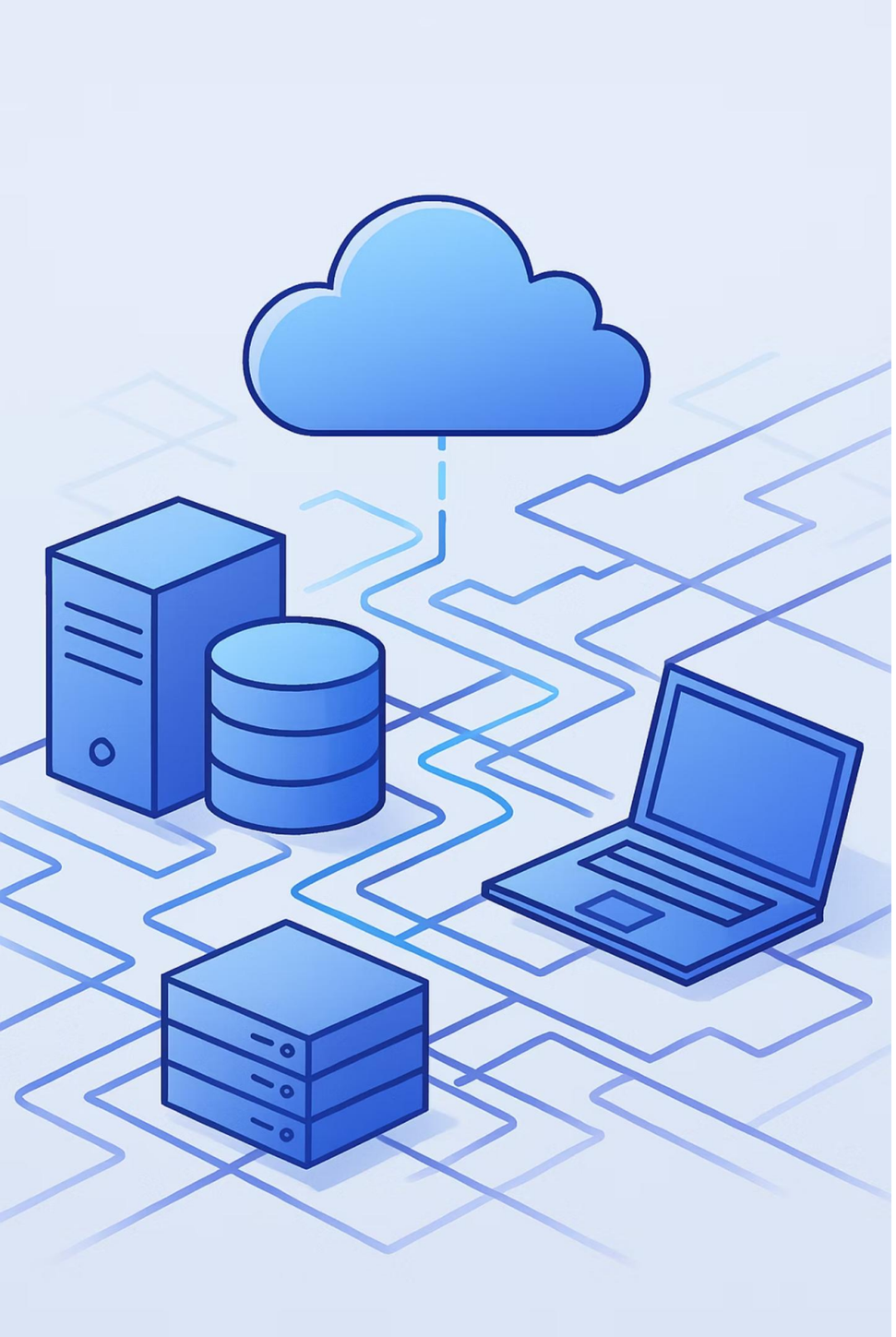


Golden Paths, Service Creation, CI/CD and AI

Today, we'll explore crucial components of modern platform engineering: Golden Paths for streamlined development, service creation patterns, CI/CD as a service, and how AI is transforming template creation and selection.

Building on yesterday's foundation, we'll examine how these technical components create an integrated developer experience that balances autonomy with governance through infrastructure abstractions and self-service capabilities.

What Are Golden Paths?

Definition

Golden Paths are technically defined, opinionated workflows that enable development teams to build, test, and deploy services efficiently using platform-approved tools and infrastructure patterns.

They represent standardized implementation paths that are fully maintained and supported by platform engineering teams, with embedded best practices for security, compliance, and operational excellence.

Core Components

- Service templates with preconfigured boilerplate
- Automated pipeline configurations
- Infrastructure-as-code modules
- Embedded security controls
- Monitoring and observability defaults

Why Golden Paths Matter

Accelerated Delivery

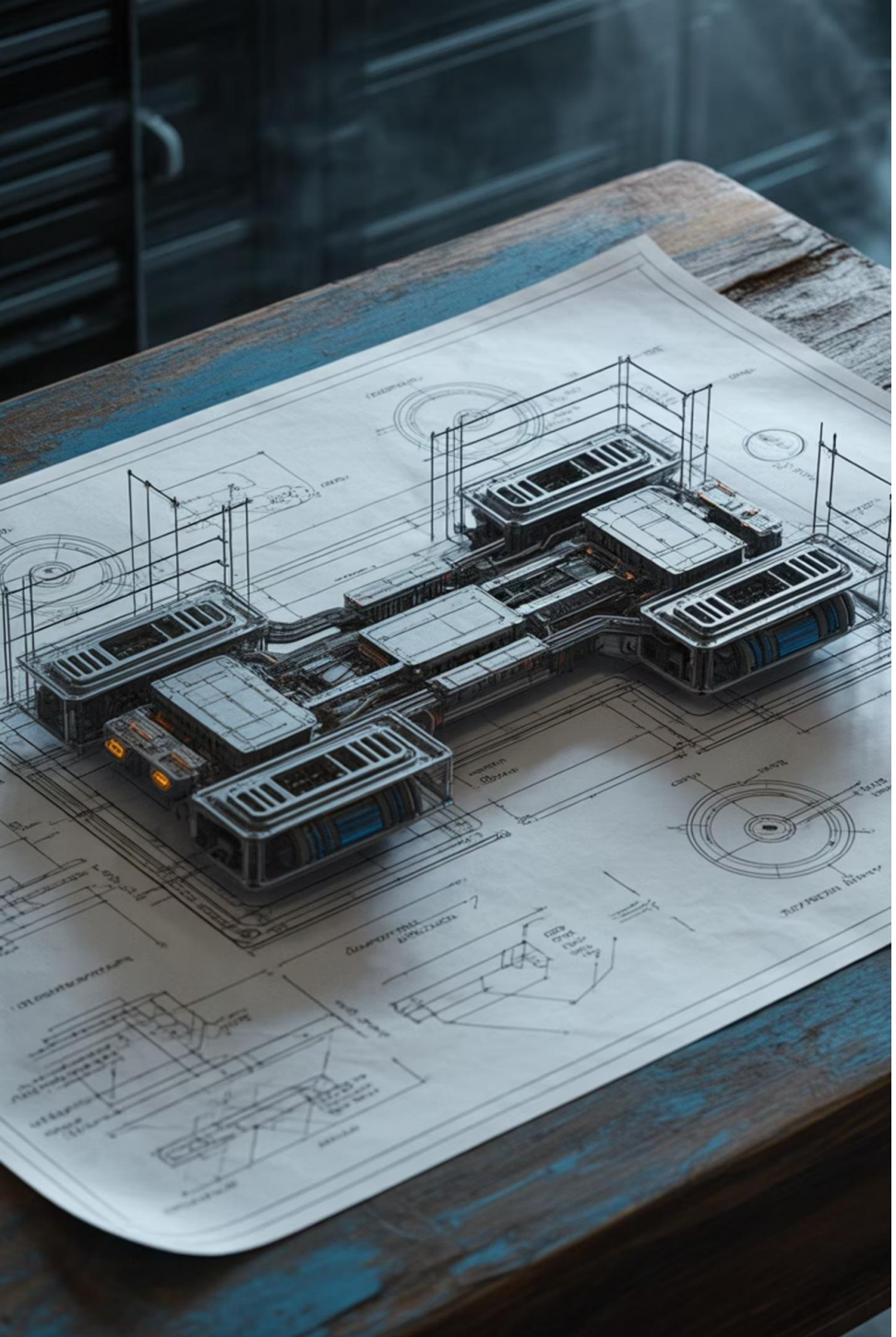
Reduces time-to-production by eliminating redundant decision-making and configuration tasks. Our measurements show 60-80% reduction in initial service setup time with properly implemented Golden Paths.

Built-in Compliance

Enforces security and regulatory controls by default, reducing the risk of non-compliant deployments. Security scanning, policy enforcement, and audit trails are incorporated automatically.

Cognitive Load Reduction

Engineers focus on business logic rather than infrastructure decisions. Technical cognitive load is measurably reduced by eliminating 90+ decisions per service implementation.



How to Design Golden Paths

Identify Common Patterns

Analyze existing workflows to identify repetitive operations and decision points that can be standardized. Focus on technology patterns with high reuse potential across multiple teams.

Create Technical Templates

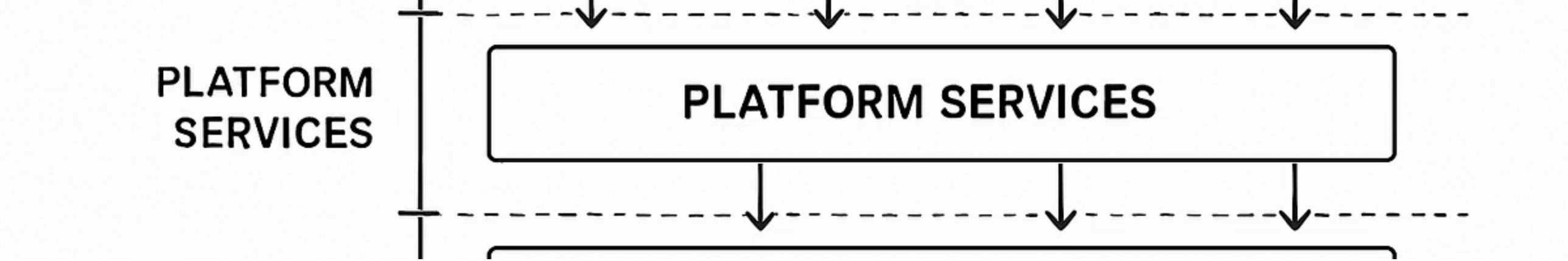
Develop programmatic templates with sensible defaults and limited configuration options. Implement as code using template engines that support inheritance and composition.

Embed Governance Controls

Incorporate policy-as-code, security checks, and compliance requirements directly into the templates and associated CI/CD pipelines.

Establish Feedback Mechanisms

Implement telemetry to measure Golden Path usage, performance, and identify friction points. Create technical channels for improvement suggestions.



Role of Abstraction Layers

Developer Interface Layer

- Self-service portals
- CLI tools with templates
- IDE plugins and extensions
- API documentation hubs

Orchestration Layer

- Template processing engines
- Pipeline orchestrators
- Service mesh controllers
- Policy enforcement points

Infrastructure Layer

- Infrastructure APIs
- Container orchestration
- Database provisioning
- Networking components

Effective abstraction creates clear separation of concerns while providing appropriate visibility and control at each layer. The technical interfaces between layers should be well-documented with versioning support.

Service Creation Process

1

Developer Initiates Service Creation

Using platform CLI or portal, developer provides minimal service metadata (name, type, team, etc.). Authentication and authorization verified against platform identity services.

2

Template Selection & Configuration

Platform suggests appropriate templates based on service requirements. Configuration options validated against technical constraints and organizational policies.

3

Service Scaffolding Generation

Template engine executes, generating repository structure with infrastructure code, CI/CD configs, and service boilerplate. Includes dockerfile, helm charts, and monitoring configurations.

4

Repository Provisioning

New repository created in version control system with proper permissions, branch protection rules, and initial commit containing generated artifacts.

5

Pipeline Configuration

CI/CD pipelines automatically configured with appropriate stages for the service type. Secrets and environment variables injected securely.

Cookiecutter and Yeoman Templates

Cookiecutter

Python-based templating system using Jinja2 for text rendering. Suitable for multi-language projects with static scaffolding needs.

```
cookiecutter-service/├── {{cookiecutter.name}}/│   ├── src/│   ├── tests/│   ├── Dockerfile│   ├── .gitlab-ci.yml│   └── README.md└── cookiecutter.json
```

Strong for simple file-based templates with variable substitution. Less interactive but excellent for GitOps workflows and CI automation.

Yeoman

JavaScript-based generator ecosystem with rich interactive capabilities. Ideal for web applications and JavaScript-heavy projects.

```
generator-service/├── generators/│   └── app/│   ├── index.js│       └── templates/├── package.json└── .yo-rc.json
```

Excels at complex, multi-step scaffolding with conditional logic and developer prompts. Provides greater extensibility through its plugin architecture.

GitOps & CI/CD as a Service – Overview

Git as Single Source of Truth

All configuration, infrastructure code, and application code stored in Git repositories. Changes to production systems always flow through version control, creating complete audit trail.

Pipeline Enforcement

CI/CD pipelines validate changes against policies before promotion to production branches. Includes security scanning, compliance checks, and performance testing.

Declarative System State

Infrastructure and application deployments defined as declarative manifests. Desired state in Git automatically reconciled with actual state in runtime environments by platform controllers.

Automated Reconciliation

Platform operators like Flux, ArgoCD, or custom controllers continuously monitor Git repositories and apply changes to target environments, handling drift detection and correction.

CI/CD Tooling Comparison

Tool	Integration	Scalability	Configuration	Best For
GitHub Actions	Tight GitHub integration	Good for small/medium	YAML in repository	GitHub-native projects, OSS
GitLab CI	GitLab ecosystem	Excellent scaling	.gitlab-ci.yml	End-to-end DevSecOps
Jenkins	Universal plugins	Highly scalable	Jenkinsfile (groovy)	Complex workflows, legacy
Tekton	Kubernetes-native	Cloud-scale	Custom Resources	Cloud-native platforms
CircleCI	Multi-cloud, agnostic	Good scaling	config.yml	Fast setup, SaaS model

When implementing CI/CD as a service, consider defining a standardized pipeline abstraction layer that provides consistent developer experience regardless of the underlying implementation tool.

CI/CD Pipeline Capabilities

Code Quality & Security

- Static Application Security Testing (SAST)
- Software Composition Analysis (SCA)
- Secret scanning
- OWASP dependency checks

Testing & Validation

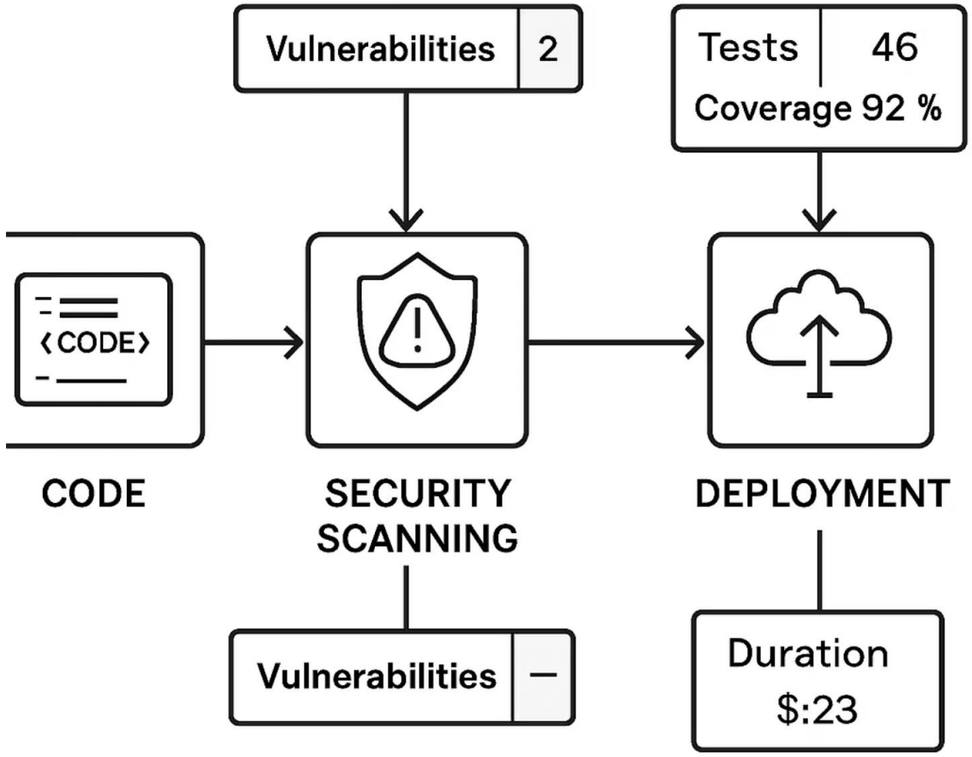
- Unit, integration, end-to-end tests
- Contract testing for APIs
- Infrastructure validation
- Policy compliance verification

Artifact Generation

- Container image building
- Software Bill of Materials (SBOM)
- Signed provenance metadata
- Version tagging and registry push

Deployment Orchestration

- Progressive delivery (Canary/Blue-Green)
- Automated rollbacks
- Environment promotion
- Post-deployment verification



AI in Service Creation

AI Capabilities in Platform Engineering

Modern AI systems can analyze requirements, context, and historical patterns to suggest appropriate templates and configurations for new services. They reduce cognitive load by handling complex decision trees that previously required platform engineering expertise.

Large Language Models (LLMs) fine-tuned on organization-specific patterns can provide intelligent scaffolding recommendations with contextual reasoning about technical constraints and organizational standards.

Implementation Approaches

- Natural language service description to template mapping
- Context-aware CLI with intelligent defaults
- Repository analysis for appropriate template suggestions
- Configuration validation with explanation capabilities
- Dynamic template customization based on requirements
- Adaptive scaffolding that evolves with usage patterns

AI-assisted Template Suggestion – Example

1

Developer Initiates Service Creation

```
$ platform create service --ai-assist
```

Platform CLI invokes AI agent to guide the template selection process.

2

AI Collects Requirements

```
AI: What type of service are you building?Dev: A RESTful API for customer dataAI: Will it need a database?Dev: Yes, PostgreSQL
```

3

Context Analysis

AI analyzes team's existing services, recent patterns, and organizational standards. It identifies related services and their architecture to inform recommendations.

4

Template Recommendation

```
AI: I recommend the 'api-with-postgres' template. It includes:- Express.js API framework- PostgreSQL client with migrations- OpenAPI documentation- Required security controls
```

5

Configuration & Generation

AI suggests sensible defaults for all required configuration fields, explains technical tradeoffs, and generates the complete service scaffold.

Golden Paths in MVP Lifecycle

Ideation & Design

Golden Paths provide architecture templates and technical decision frameworks. AI can recommend appropriate architectural patterns based on requirements analysis and similar successful projects.

Iteration

Feedback mechanisms and telemetry capture usage patterns to improve templates. Golden Paths evolve based on operational experience and changing requirements.

Development

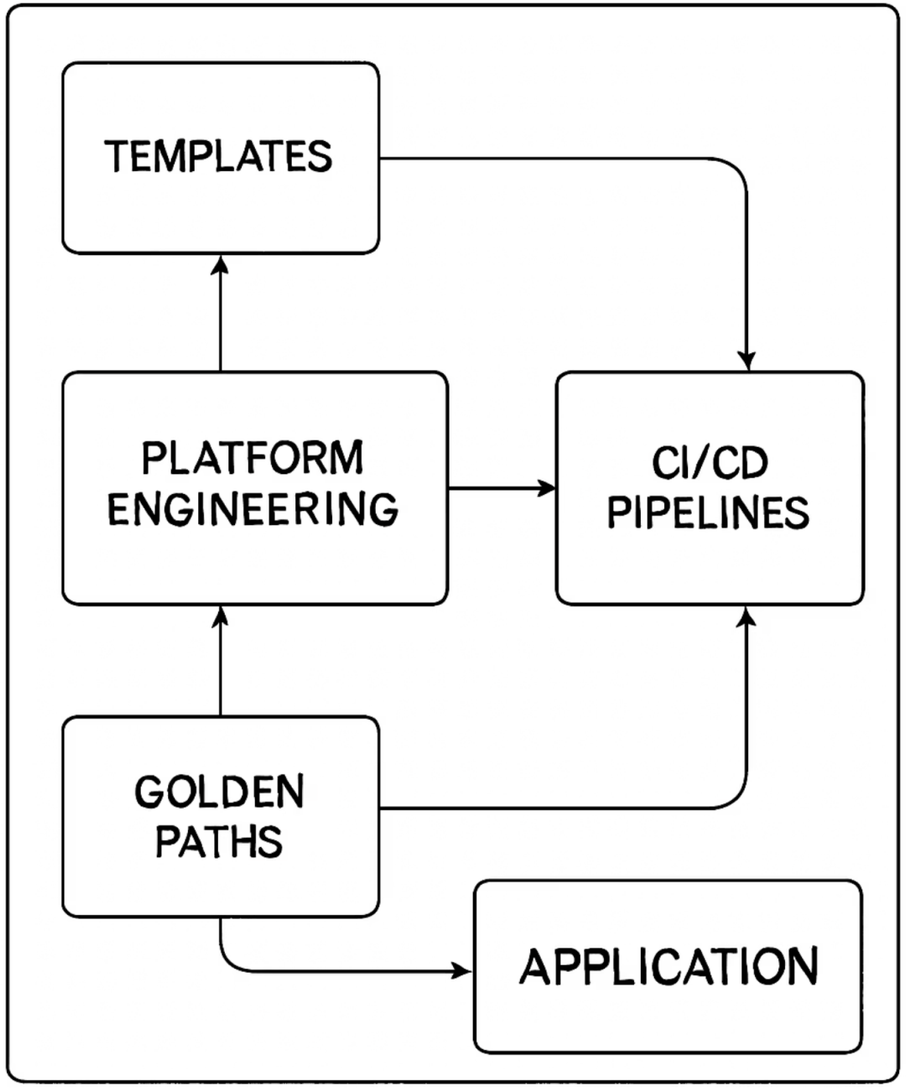
Service scaffolding with predefined capabilities, integrated development environments, and standardized build processes. Templates include testing frameworks and quality gates.

Deployment

Automated pipelines for continuous deployment with progressive delivery patterns. Infrastructure as code templates ensure consistent environments with proper security controls.

Operations

Built-in observability, monitoring dashboards, and alerting configurations. Runbooks and incident response workflows integrated with platform tools.



Recap of Key Concepts

Golden Paths

Opinionated, standardized workflows that reduce cognitive load and accelerate development by embedding platform engineering expertise into reusable patterns. They provide guardrails while enabling developer autonomy.

Template Generation

Programmatic scaffolding using tools like Cookiecutter and Yeoman to create consistent service foundations. Templates encode architectural decisions, security controls, and operational best practices.

GitOps CI/CD

Declarative infrastructure and application definitions versioned in Git, with automated pipelines for validation, testing, and deployment. CI/CD as a service provides standardized pipeline capabilities.

AI-assisted Scaffolding

Intelligent assistance for template selection and configuration, reducing cognitive load on developers while ensuring alignment with organizational standards and best practices.