

Platform Engineering Essentials

Empowering Developer Innovation Through Self-Service Platforms

WRITTEN BY **APOSTOLOS GIANNAKIDIS**

PRINCIPAL PRODUCT SECURITY ENGINEER, MICROSOFT

CONTRIBUTIONS FROM **KELLYN GORMAN**

CONTENTS

- About Platform Engineering
 - Key Components of Platform Engineering
- Core Platform Engineering Capabilities
 - Self-Service Developer Portal
 - Repository Templates
 - Software/Service Catalog
 - Documentation Repository
 - GenAI Capabilities "as a Service"
- Conclusion and Additional Resources

Platform engineering is a discipline focused on creating and managing development platforms that significantly enhance software delivery processes. At its core, **platform engineering** aims to establish secure environments, automated and self-service tools, and streamlined workflows that empower development teams to write, test, and deploy applications effectively and consistently across various settings without worrying about operational complexities. As technology evolves, the integration of automation, security, and AI into these platforms becomes crucial, transforming how software is built and maintained.

This Refcard will cover the strategic value of platform engineering, explore best practices and tools, demonstrate how to drive organizational outcomes and align platform engineering with business goals, and prepare the reader for future trends. The goal is to equip the reader with the knowledge and tools to harness platform engineering as a powerful enabler of modern software delivery, setting the stage for more agile, secure, and efficient development processes.

ABOUT PLATFORM ENGINEERING

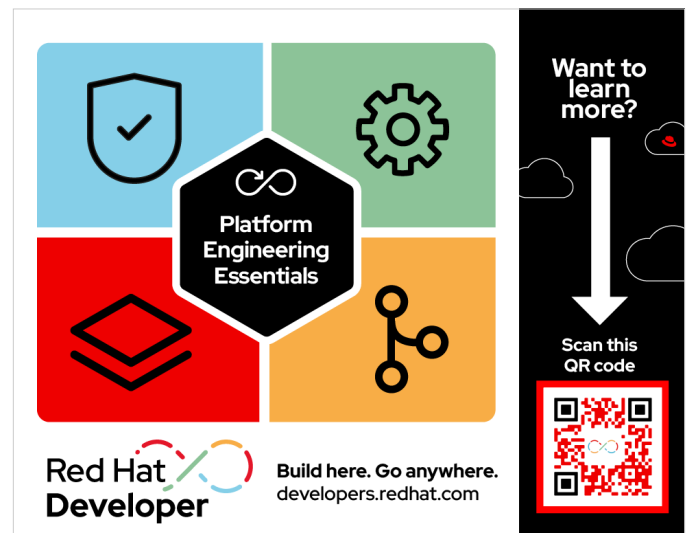
While the DevOps movement has improved development speed and enhanced collaboration between technical teams, the rapid growth in the number of development teams in large-scale DevOps environments, coupled with an increase of complex services and tools, has resulted in a heavy cognitive load. This burden reduces productivity and often leads to burnout, thus resulting in inefficiencies, duplication, higher maintenance costs, reduced adaptability, and increased security risks. Platform engineering addresses these challenges by reducing cognitive load and improving productivity through optimized developer experiences and streamlined operations, often called golden paths.

Platform engineering is the practice of designing, building, and managing unified [internal development platforms](#) that standardize and streamline software delivery processes across an organization.

IDPs achieve this by providing a curated set of tools, environments, and services that are easily discoverable, readily available, secure, scalable, and provided in a self-service manner. This empowers development teams to build, deploy, and maintain applications efficiently and consistently across different environments. By doing so, platform engineering:

- Accelerates developer productivity and delivery
- Improves the overall development experience
- Significantly reduces the risk of security, regulatory, and functional issues throughout the SDLC

A **platform engineering team** is responsible for *creating the platform* following the "Platform-as-a-Product" approach. The platform offers standardized tools, automated workflows, and golden path templates, as well as provides necessary abstractions to ensure scalable, secure, and efficient application deployments.



The infographic features a central black hexagon with a white infinity symbol and the text "Platform Engineering Essentials". Surrounding this central hexagon are four colored squares: a blue square with a shield icon (top-left), a green square with a gear icon (top-right), a red square with a stack of layers icon (bottom-left), and an orange square with a network node icon (bottom-right). Below the central hexagon, the "Red Hat Developer" logo is displayed on the left, and the text "Build here. Go anywhere. developers.redhat.com" is on the right. To the right of the infographic, a vertical black bar contains the text "Want to learn more?" at the top, followed by a large white downward-pointing arrow, and then "Scan this QR code" above a QR code.



Build here.
Go anywhere.

developers.redhat.com

Platform Engineering Essentials

Improving productivity for all development teams



Want to learn more?
Scan the QR code



While **development teams** are the *primary users of this platform*, it also benefits other teams — such as site reliability engineering (SRE), security, product support, and FinOps — enabling collaboration and consistency across the organization.

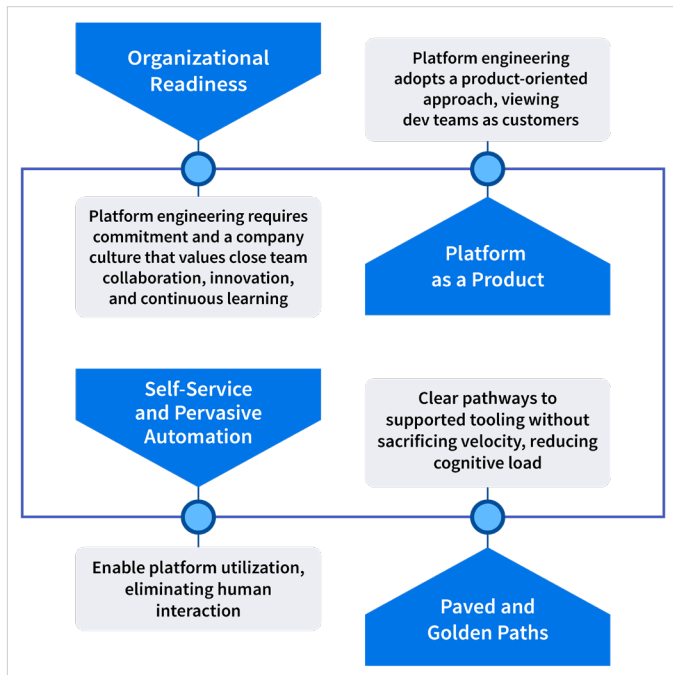
Table 1 outlines the key benefits and objectives of platform engineering:

Table 1: Platform engineering advantages and objectives

BENEFITS	OBJECTIVES
Improves the developer experience	<ul style="list-style-type: none"> Reduce context switching and maintenance overhead Expedite product delivery Enhance technology adoption Reduce developer friction
Enhances development team efficiency	<ul style="list-style-type: none"> Reduce cognitive load Streamline development processes Enhance productivity
Enhances infrastructure and operations team efficiency	<ul style="list-style-type: none"> Reduce number of tickets Increase capacity for strategic initiatives Reduce change failure rate
Strengthens security and compliance	<ul style="list-style-type: none"> Improve security posture and compliance across the organization Minimize vulnerabilities, risks, and recovery times Increase efficiency, transparency, and trust

Figure 1 outlines a set of principles to help guide the design and implementation of platform engineering practices.

Figure 1: Core principles of effective platform engineering



Challenges faced by platform engineering include:

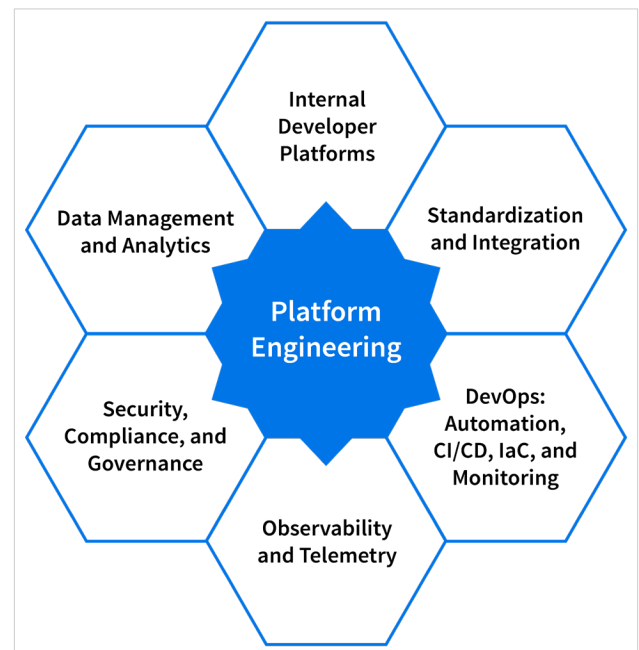
- **Rapid technological advancement.** Staying current with the constant evolution of tools and technologies can be overwhelming. Platform engineers need to continuously learn and adapt to new advancements to ensure their platforms remain relevant and competitive.

- **Infrastructure complexity.** Managing the growing complexity of infrastructure, from distributed systems to containerized environments, brings technical challenges in troubleshooting, scaling, maintenance, and operational efficiency.
- **Integrating diverse tools and technologies.** Ensuring system compatibility, managing the complexity of multiple integrations, and keeping up with the latest advancements are challenges introduced by the platform's diverse tools.
- **Managing security and compliance.** Platform engineers must continuously implement security controls to ensure comprehensive coverage against ever-evolving threats across all environments, monitor systems, and respond to incidents while maintaining compliance with evolving industry regulations.
- **Automating complex workflows.** Identifying and automating intricate processes and toolchains without introducing errors while maintaining a cohesive and efficient platform is a key challenge.
- **Enabling cross-functional collaboration.** Getting development, operations, security, and product management teams to work together smoothly can be challenging. Differences in their tools and workflows often cause collaboration issues, making it harder to build a unified platform.

KEY COMPONENTS OF PLATFORM ENGINEERING

As platform engineering continues to evolve, its key components are still taking shape. Nevertheless, the following fundamental elements are essential for any successful platform.

Figure 2: Core components of platform engineering



INTERNAL DEVELOPER PLATFORMS

An **internal developer platform** (IDP) is the cornerstone of platform engineering, providing a central collection of tools, services, and automated workflows that cover the operational necessities of the

entire SDLC to support the rapid development and delivery of software products across the organization. By abstracting the complexities of application configuration and infrastructure management, IDPs provide a standardized interface for managing application lifecycles without direct interaction with underlying systems, while also preserving the useful and differentiated capabilities of the underlying core services.

A fundamental property of IDPs is their *self-service* capability, allowing development teams to independently access the resources they need with minimal friction and without human intervention. For example: Instead of waiting for operational teams to provision infrastructure or resolve tickets, developers can easily spin up cloud environments, trigger CI/CD pipelines, implement rollbacks, access logs, build their code, and manage build artifacts from a single API or GUI. This self-service capability reduces bottlenecks, enhances productivity, and promotes standardized environments and workflows across development, testing, and production.

STANDARDIZATION AND INTEGRATION

In the platform engineering discipline, **standardized environments** play a fundamental role: By ensuring uniformity in development, testing, and production environments, standardization reduces the common issue of "it works on my machine." Uniformity also:

- Facilitates the scaling of applications
- Provides predictable and reproducible environments
- Simplifies maintenance and upgrades
- Reduces downtime
- Improves reliability

A unified platform enhances collaboration among teams as everyone works within the same framework and set of tools, leading to improved productivity. In many cases, standardization involves defining golden paths.

Integrating with underlying services and abstracting their complexities ensures that developers do not need to interact directly with the intricate details of infrastructure and application configuration. Instead, they can work with a standardized interface that provides consistent and reliable access to the necessary resources. This abstraction layer hides the complexity of the underlying systems while still leveraging their powerful capabilities. As a result, it reduces the cognitive load on development teams, minimizes errors, and accelerates the development lifecycle, ultimately leading to more efficient and effective software delivery.

DEVOPS: AUTOMATION, CI/CD, IAC, AND MONITORING

Building upon the core principles of **DevOps**, platform engineering has emerged as the key strategy for scaling its impact, thus prioritizing collaboration, automation, and continuous delivery. Platform engineering takes DevOps to the next level by providing a more structured and scalable approach to delivering value to the business.

The principles of DevOps and DevSecOps are at the core of IDPs. The goal is to eliminate all manual intervention, enabling faster and more reliable deployment of applications. By automating repetitive tasks like infrastructure provisioning through Infrastructure as Code (IaC) and automated pipelines, platform engineering minimizes errors, improves consistency, and accelerates software development cycles, thereby aligning with the DevOps goals of agility and speed.

Moreover, DevOps practices in platform engineering enhance cross-team collaboration between developers, operations, and other stakeholders. With self-service capabilities and standardized workflows provided by IDPs, platform engineering empowers teams to deploy, monitor, and maintain applications independently. This not only improves the developer experience but also ensures security, governance, and compliance across different environments.

The combination of DevOps methodologies and platform engineering allows organizations to scale their software delivery process while maintaining flexibility, enabling them to respond quickly to changing business needs and market demands.

INFRASTRUCTURE AS CODE

IaC is a cornerstone of platform engineering: By removing manual processes and ensuring consistency across environments, IaC streamlines DevOps pipelines and reduces the risk of configuration drift. This not only improves security but also enables organizations to provision identical environments repeatedly, a fundamental aspect of platform engineering. The choice of an IaC framework is critical for a successful platform engineering strategy because it dictates how infrastructure is defined, tested, and deployed, influencing its usability, reusability, and scalability.

MONITORING AND LOGGING

Monitoring and logging provide visibility into application performance, security, and health. By integrating robust monitoring and logging solutions, platform engineering teams can maintain high standards of performance, high availability, security, and user satisfaction, while also facilitating proactive monitoring, rapid troubleshooting, and continuous optimization.

Platform engineering unifies monitoring tools and establishes a consistent approach to monitoring the entire technology stack. Platforms provide centralized dashboards, aggregating data from all services, offering a comprehensive overview of system health, and facilitating rapid issue resolution.

OBSERVABILITY AND TELEMETRY

Observability in platform engineering refers to the ability to understand the internal state of a system by analyzing its logs, metrics, and traces. This allows teams to monitor the performance and health of both the platform and the applications running on it. Telemetry plays a crucial role by collecting real-time data that provides visibility into system behavior, enabling development teams to proactively identify and resolve issues before they escalate.

In platform engineering, observability is a shared responsibility that fosters assisted ownership. Platform and application teams work together to ensure observability, with the platform team providing the essential infrastructure, templates, and pre-built dashboards to help development teams easily monitor their services and applications with minimal effort. These dashboards offer actionable insights and proactive recommendations, enabling teams to maintain platform stability and performance.

SECURITY, COMPLIANCE, AND GOVERNANCE

By adopting a **security-first approach** to platform design, organizations reduce the cognitive load needed to comply with security requirements and policies. Embedding security measures such as encryption, access control, and vulnerability management from the outset ensures that all services and apps consistently follow standardized security policies, making compliance seamless and efficient. DevSecOps integrates security tools directly into CI/CD pipelines, such as static application security testing (SAST) and software composition analysis (SCA), enabling continuous monitoring and vulnerability detection without hindering development speed. By providing development teams with built-in security gates, platform engineering ensures that robust security practices are maintained while preserving the efficiency and flow of the development process.

Compliance and governance are enforced through automated checks and audits, ensuring adherence to regulatory standards like [GDPR](#), [HIPAA](#), or [PCI-DSS](#). The platform implements secure configurations, compliance automation, and access management by default, ensuring that security and governance policies are consistently applied across the organization.

Features such as single sign-on, role-based access control (RBAC), network security, image scanning, and [Open Policy Agent](#) enforce granular policies at the resource level, creating a secure and compliant environment that scales efficiently with organizational needs.

DATA MANAGEMENT AND ANALYTICS

The platform handles data ingestion, transformation, and exposure, enabling teams to efficiently process raw data and turn it into actionable insights. Platform engineering has advanced data platforms by integrating IDPs, data fabric, and the "data mesh" principle.

Data fabric facilitates intelligent orchestration and automation, enabling active, metadata-driven integration and analysis of diverse data sources. The **data mesh** principle decentralizes data ownership, giving domain-specific teams control over their data pipelines. Unlike traditional centralized architectures, the data mesh shifts responsibility to the teams that generate and use the data, turning them into data "product" owners. This encourages cross-departmental collaboration and allows teams to autonomously manage workflows, leading to more efficient data operations.

The combination of data fabric and data mesh provides enhanced self-service capabilities, enabling secure, frictionless data management

and exchange while simplifying processes and accelerating innovation. Data mesh defines the governance and rules for data sharing, which data fabric then enforces, ensuring smooth implementation across the organization.

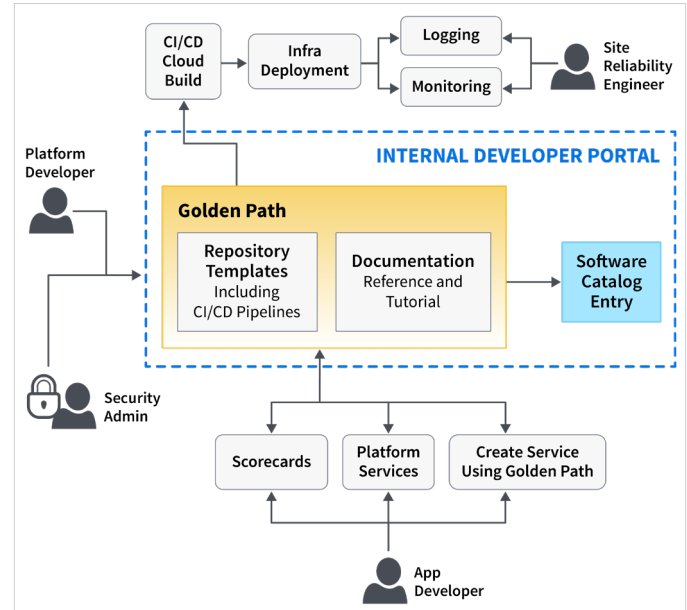
In platform engineering, data is increasingly treated as a *product* rather than a byproduct. This elevates its importance within the organization by applying formal definitions of ownership, schema, quality, and security.

CORE PLATFORM ENGINEERING CAPABILITIES

Each engineering platform will evolve uniquely based on the organization's specific needs. However, most internal engineering platforms incorporate several fundamental capability domains, including those below:

- An internal developer portal
- Repository templates
- Paved and golden paths
- A software/service catalog
- A documentation repository
- Scorecards
- GenAI capabilities "as a Service"

Figure 3: Overview of platform engineering capabilities



SELF-SERVICE DEVELOPER PORTAL

Developer portals streamline operations with internal developer platforms (IDPs), making it easier for teams to discover and leverage platform resources and services effectively. According to Gartner, by 2026, [75% of organizations with platform engineering teams](#) will offer developer portals to enhance the developer experience and accelerate innovation.

Developer portals typically consist of the following key components:

- Universal software and service catalog
- Software templates based on existing practices
- Technical auto-documentation
- Access management via RBAC

Developer portals offer self-service and automation capabilities, allowing development teams to independently perform routine tasks such as automatically provisioning environments, deploying applications, discovering services, and requesting permissions. Tools embedded in the portal can automate tasks like triggering alerts, terminating temporary environments, and managing permissions, therefore ensuring consistency and improving the efficiency of development workflows. Portals also provide platform engineering and development teams with access to dashboards that display platform performance, status, and adoption metrics.

DEVELOPER PORTAL EXAMPLE: BACKSTAGE.IO

One of the earliest and most widely adopted developer portals is [Backstage.io](https://backstage.io), an open-source framework originally developed by Spotify. Rather than being a standalone portal, Backstage serves as a framework that allows platform engineers to create customized developer portals. This flexibility is due to Backstage's plugin-based architecture, which makes it highly extensible and adaptable for various use cases, allowing development teams to customize their experience.

However, Backstage lacks enterprise-level features such as RBAC, dedicated customer support, extended lifecycle management, certifications, and support for legacy versions. This gap is filled by enterprise-grade internal developer portals, which either extend Backstage or offer more comprehensive enterprise solutions. These platforms provide advanced features like RBAC, integration with container platforms, dynamic and verified plugins, orchestration for configuration and infrastructure, data aggregation, and scalable infrastructure.

REPOSITORY TEMPLATES

Repository templates allow development teams to share boilerplate code across codebases. This capability enables efficient project bootstrapping with preferred tools and directory structures, significantly reducing manual configuration and compatibility issues. By standardizing the initial set-up process, repository templates ensure consistency across projects, making it easier for developers to transition between different codebases and maintain a uniform development environment.

Platforms provide a centralized way to store these templates, offering a streamlined method for accessing and utilizing them in each repository's pipelines. This centralization simplifies template management and enhances collaboration among development teams. By storing repository templates and reusable workflows in a centralized repository, they can be used across multiple repositories

while being managed in a central location. This approach ensures that development teams consistently follow best practices and organizational standards, improving efficiency and reducing potential errors. Additionally, it provides predefined structures for bug reports, feature requests, and discussions, further streamlining the development process. Finally, repository templates are the foundation upon which golden paths are built.

PAVED AND GOLDEN PATHS

One of the key ways platform engineering reduces cognitive load and complexity is via self-service templates for common tasks that streamline the development process, called paved paths or golden paths. **Golden paths** are designed to provide a single, clear method to accomplish specific tasks so that teams don't have to reinvent the wheel. They reduce the cognitive load on developers by using abstractions and comprehensive documentation. Essentially, golden paths are configurable, extensible templates that offer transparency, making it easy for developers to understand, customize, and add additional capabilities if required.

In practice, golden paths typically include clear instructions for developers, a clearly defined audience, a repository template, platform integrations with services and tooling, templated CI/CD pipelines, deployment manifests, observability capabilities, and IaC configurations for both staging and production environments. The self-service nature of golden paths, along with the internal developer portal, allow development teams to discover and use them without the need for a ticketing system.

Platform engineering teams build, document, and maintain the golden path templates. They also set KPIs and quantitative goals and create dashboards to track golden path adoption within the organization.

EXAMPLE GOLDEN PATH USE CASE

Let's see a typical example of how golden paths can help development teams. The workflow starts with the platform engineering team creating a golden path for development teams who need to deploy microservices using Kubernetes. This golden path includes a pre-configured [Kubernetes Deployment](#) template integrated with the company's existing managed Kubernetes cloud service, along with automated security policies and observability tools like [Prometheus](#) for monitoring and [Grafana](#) for visualization.

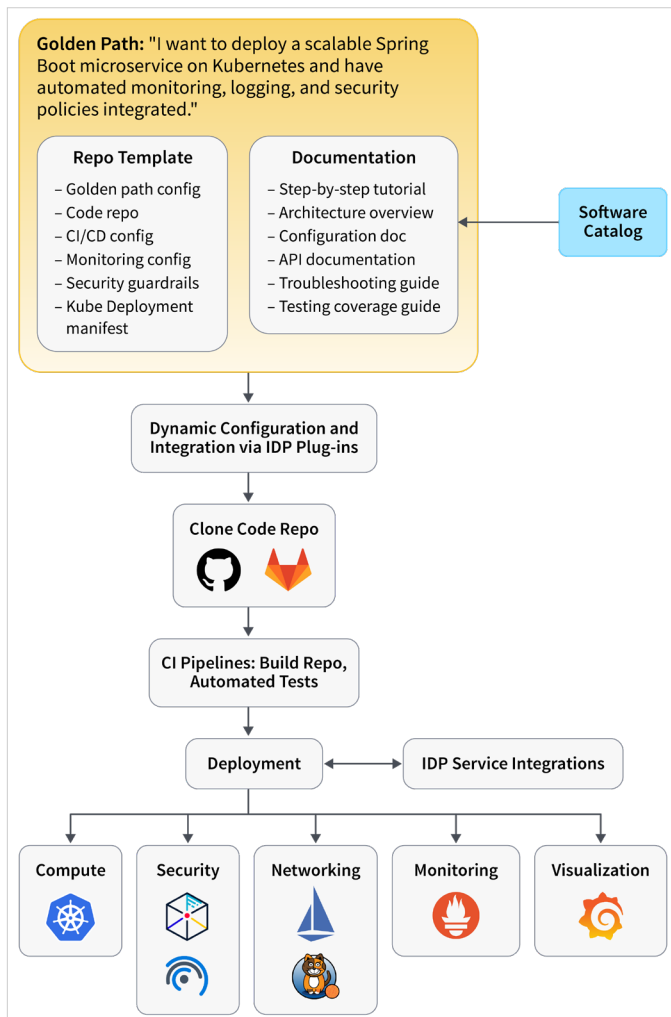
Each golden path is typically associated with an intent-based statement, and for this particular use case, the statement could be:

"I want to deploy a scalable Spring Boot microservice on Kubernetes and have automated monitoring, logging, and security policies integrated."

A development team working on a new microservice can use the internal developer portal and select the specific golden path template, which automatically provisions the infrastructure (e.g., staging and production environments) and applies the necessary CI/CD pipeline for continuous deployment. The golden path applies built-in best

practices for security and compliance, such as RBAC, container image scanning, and static code analysis. By following this predefined golden path, the development team can reduce the time spent on configuring environments and troubleshooting infrastructure issues, allowing them to focus on building and improving the microservice itself.

Figure 4: Abstract illustration of the golden path example use case



SOFTWARE/SERVICE CATALOG

Service catalogs help organizations avoid duplicated efforts by making projects and resources visible across teams. As projects, products, and teams evolve, projects can become orphaned. Inventories help address these issues by acting as a single source of truth. Inventories improve security, promote reuse, and make discovery easier. An inventory is a tool or system used to track, manage, and organize an organization's technical assets, including code, templates, APIs, containers, virtual machines (VMs), and team permissions. The open-source portal framework, Backstage.io, refers to this capability as a software catalog while other products may use terms like service or software inventory.

Not keeping track of assets leads to technical sprawl because existing resources are not easily discoverable. For example, an organization might have numerous containers and VM instances running without knowing which ones can be safely deleted. Proper tagging and tracking

can help identify the owners or teams responsible for these assets, ensuring decisions about the resource's lifecycle are well informed.

Simple software catalog implementations (e.g., wiki pages) can help track relationships between assets, but they quickly become outdated and require significant manual effort. In platform engineering, service catalogs are typically maintained dynamically through platform integrations.

SOFTWARE CATALOG EXAMPLE: BACKSTAGE.IO

In Backstage, the catalog is kept up to date via a data ingestion process. This process involves using entity providers or processors that fetch raw entity data from external sources. When a developer creates Kubernetes resources using one of the available golden paths in Backstage, the new resources automatically generate entries in the software catalog. This is achieved through the [Backstage entity provider plugin](#), which reads Kubernetes objects as Backstage entities. This dynamic maintenance ensures that the catalog remains current, provides a holistic view of the infrastructure, and reduces the manual effort required to keep it updated.

DOCUMENTATION REPOSITORY

While development teams write documentation as part of their operations, they often face challenges, particularly in the areas of discoverability, completeness, and consistency. Platform engineering addresses these issues by integrating the "Docs-like-Code" or "Docs-as-Code" approach into the IDP. Engineers write documentation in Markdown, or sometimes AsciiDoc files, which reside alongside their code, commonly referred to as TechDocs. Repository templates provide everything needed to set up a documentation site with minimal configuration, resulting in a well-organized and easy-to-find doc site within the portal.

TechDocs is a Docs-as-Code solution (e.g., Backstage's [TechDocs](#)) that retrieves documentation from service repositories and displays it in an easy-to-read and searchable format. This approach removes guesswork for developers as the IDP automatically pulls compatible documentation files (e.g., Markdown) from the repository, generates the HTML pages, and displays them on the appropriate service page. IDPs also provide powerful search functionality, making it easy to find and access documentation.

The documentation generation process typically involves:

1. A CI/CD pipeline that fetches Markdown files from the source code hosting provider (e.g., [GitHub](#), [GitLab](#)).
2. The TechDocs generator then builds static HTML pages, including stylesheets (CSS files) and scripts (e.g., JavaScript), using static site generators like [MkDocs](#).
3. The TechDocs publisher uploads the generated files to a cloud storage, which can be any of the public cloud providers.

Managing access to documentation in cloud storage is very important for the security of the developer portal. As a best security practice, IDPs restrict access to the cloud storage and allow only the IDPs'

documentation component to fetch files. Authorized IDP users should have read-only access to the generated documentation while the CI/CD pipeline requires write permissions to publish the generated site files. Additionally, to protect against XSS attacks, TechDocs use XSS sanitizer libraries, such as [DOMPurify](#), to sanitize the generated HTML.

SCORECARDS

Scorecards have become a core capability within internal developer platforms. They establish engineering standards and help ensure that software meets expectations around security, reliability, operations, production readiness, compliance, and deployment. By defining standards such as production readiness and development quality, scorecards eliminate the need for custom scripts and spreadsheets, enabling development and operations teams to track metrics based on service properties effectively.

The essence of scorecards lies in their checks, which are configured and evaluated against the services to produce a score. Each scorecard consists of a set of rules, with each rule defining one or more conditions that must be met. This structured approach allows organizations to establish meaningful standards as well as helps developers improve and measure their progress.

By ingesting data via the software catalog and integrations with services, scorecards enable the programmatic evaluation of these standards, providing teams with the insights and motivation needed to achieve better compliance with policies. Checks run against source code and the APIs exposed by services and tools, identifying components that do not meet expectations and reporting these issues to the appropriate personnel.

Scorecards use the checks to evaluate the maturity, production readiness, and engineering quality of any entity in the software catalog. For instance, they can determine whether:

- A service has an on-call defined
- Grafana is set up for the Kubernetes cluster
- The relationships of certain entities are empty
- Services are ready for production deployment

Scorecards can verify:

- Presence of runbooks
- Dashboards
- Logs
- On-call escalation policies
- Monitoring/alerting setups
- Accountable owners

Scorecards can also group checks into higher-level abstractions, such as "Security OWASP Top 10 compliance - Level 1" or "Production SRE requirements - Level 3." This makes it easier for teams to comprehend the level of maturity of their services and receive guidance on how to improve their scores. When scores are degraded, scorecards serve as an

alert mechanism. This is particularly useful when software migrations or upgrades are performed to ensure they are completed correctly. By grading all entities within the IDP, scorecards help prioritize attention and alert relevant teams when necessary. The real-time reporting of scorecards provides a clear and up-to-date picture of progress and areas needing attention, ensuring that software remains reliable, secure, and compliant with organizational standards.

GENAI CAPABILITIES "AS A SERVICE"

Although generative AI (GenAI) is not yet a core capability of IDPs, it is rapidly evolving into one. **GenAI** capabilities are giving rise to intelligent IDPs that offer GenAI-driven services. By providing AI-based copilot and companion services — such as AI-driven code generation, code review and analysis, AI-generated documentation, and AI-augmented DevSecOps — these platforms streamline SDLC workflows and boost productivity by reducing manual effort.

Another significant value of intelligent IDPs is the ability to share large language model (LLM) prompts among development teams and across different lines of business within the organization. This capability enables teams to leverage collective expertise and insights. By sharing prompts, developers can quickly access and implement best practices and innovative solutions, leading to more efficient use of LLMs and accelerated development cycles.

Also, enabling AI to access the developer environment helps platform teams create a more compelling IDP. These intelligent platforms promise a frictionless, self-service developer experience with minimal overhead and are rapidly becoming the backbone of digital transformation.

CONCLUSION

While DevOps has been instrumental in bridging the gap between development and operations, its complexity has created too much cognitive load for developers in large-scale environments. Platform engineering seeks to address this by simplifying processes, providing appropriate levels of abstraction, and establishing standardized workflows and golden paths to streamline development and operations.

By establishing standardized environments, utilizing advanced tools and automation, and implementing streamlined workflows, organizations can significantly enhance their software delivery processes. The benefits of platform engineering extend beyond improved developer productivity and experience; they also ensure robust security and compliance, which are essential in today's fast-paced and ever-evolving technological landscape.

To further your understanding and implementation of platform engineering, consider exploring the following resources:

- ["AI is Changing the Future of Platform Engineering: Are you Ready?"](#) by Aeris Ransom, Platform Engineering

CONTINUES ON NEXT PAGE

- [CNCF Platforms White Paper](#), Cloud Native Computing Foundation
- "[How Dynamic Internal Developer Platforms Boost Developer Experience and Productivity](#)" by Luca Galante, DZone
- "[Light the way ahead: Platform Engineering, Golden Paths, and the power of self-service](#)" by Darren Evans and Megan O'Keefe, Google Cloud Blog
- "[The Impact of AI and Platform Engineering on Cloud Native Evolution](#)" by Kellyn Gorman, DZone
- "[The Rise of the Platform Engineer: How to Deal With the Increasing Complexity of Software](#)" by Mirco Hering, DZone

By leveraging these resources, you'll gain the essential knowledge and practical skills needed to excel in the field of platform engineering. These resources not only cover the core principles but also provide practical guidance on mastering essential tools and technologies. With this comprehensive understanding, you'll be well equipped to contribute meaningfully to platform engineering initiatives and drive innovation more efficiently within your organization.

WRITTEN BY APOSTOLOS GIANNAKIDIS,

PRINCIPAL PRODUCT SECURITY ENGINEER, MICROSOFT



Apostolos is currently leading the Threat Modeling program at Microsoft Identity. Before joining Microsoft, he served as VP of Application Security at JP Morgan Chase and led the security strategy at Waratek. He has been acknowledged by Oracle, he is featured on Google's Vulnerability Hall of Fame, and he holds two MSc degrees in Computer Science and Cloud Computing. When not working, you might find him automating his coffee machine to brew the perfect cup of Greek coffee.



3343 Perimeter Hill Dr, Suite 100
Nashville, TN 37211
888.678.0399 | 919.678.0300

At DZone, we foster a collaborative environment that empowers developers and tech professionals to share knowledge, build skills, and solve problems through content, code, and community. We thoughtfully — and with intention — challenge the status quo and value diverse perspectives so that, as one, we can inspire positive change through technology.

Copyright © 2024 DZone. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means of electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.