Harmony in code: Unpacking DevOps and platform engineering in a cloud-native world

Learn how these disciplines can help improve developer productivity and software delivery.

Introduction

Amid the ever-evolving software development landscape, the debate surrounding DevOps vs. platform engineering has ignited, captivating the attention of professionals and enthusiasts alike. The questions arise: (1) Which side of this coin do you favor or (2) are these two domains interlinked, representing two sides of the same coin?

To demystify the intricate relationship between DevOps and platform engineering, this ebook delves into the core of these practices and explores the nuances, best practices, and tangible benefits each brings. Moreover, it navigates the challenges and complexities organizations face in adopting and optimizing these methodologies.

This ebook presents an opportunity to learn more about DevOps and platform engineering, providing a space to form your perspective about the transformative potential of these methodologies for your organization.

dynatrace

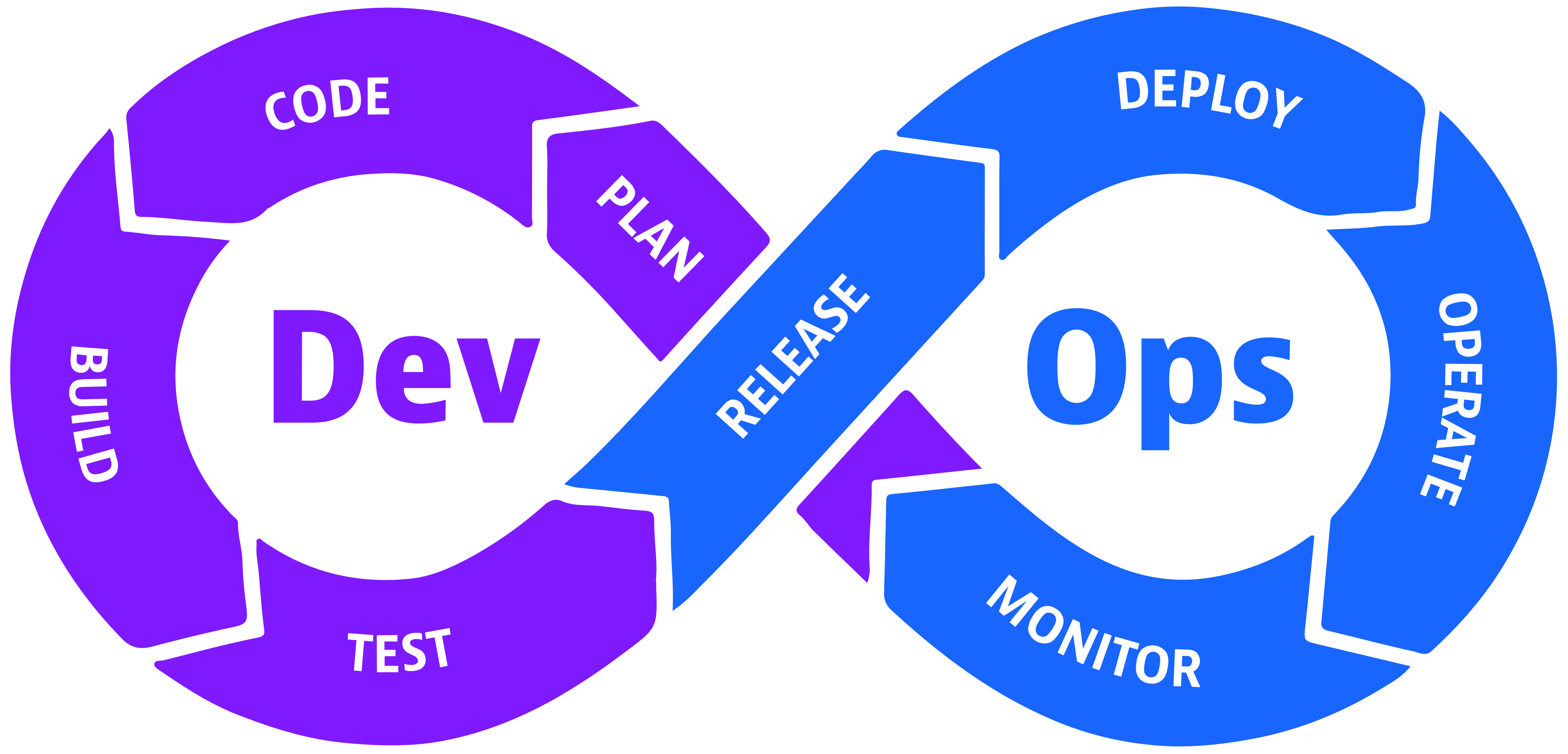
What's inside

CHAPTER ONE

What are DevOps and platform engineering?

DevOps is a flexible framework of software development practices organizations use to create and deliver software by aligning and coordinating software development efforts between two key teams: development (or “Dev") and IT operations (“Ops”).

The easiest way to understand DevOps is to visualize it as a continuous loop. Instead of discrete processes, tasks in development and operations become part of an ongoing cycle that includes building, testing, and monitoring applications and services. The DevOps methodology embodies a cultural shift that requires vision, planning, executive buy-in, and tight collaboration to successfully establish an integrated way of developing and delivering applications.



By embracing a few fundamental practices, teams can improve their efficiency and develop a deeper understanding of their workflows, toolsets, and processes to release better software faster. The following describes the basic tenets and practices that constitute a DevOps approach:

Continuous integration
Continuous integration (CI) is a software development practice in which developers regularly commit their code to a shared repository. With CI, the distributed nature of microservices architecture allows developers to own discrete, manageable chunks of code alongside individual features and work on them in parallel. Each chunk is a software artifact that teams can manage individually.

Continuous delivery
Complementing CI, continuous delivery (CD) takes artifacts that have been pushed to an artifact repository and then deploys the artifacts to multiple environments with different testing and quality assurance criteria that determine the artifacts' promotion from one stage to another.

Continuous testing and validation
Continuous testing evaluates software quality at each stage of the delivery lifecycle. Ideally using automation, continuous testing identifies poor-quality code and provides fast and continuous feedback to the development teams with the necessary information to address quality concerns.

The DevOps methodology embodies a cultural shift that requires vision, planning, executive buy-in, and tight collaboration to successfully establish an integrated way of developing and delivering applications.

Observability
Observability is key to understanding the viability of code as it progresses through the pipeline, and it provides insights into the dynamics of software as it reaches the deployed stage. With awareness of logs, metrics, traces, and other key telemetry data, teams can see where potential software issues may arise. Continual data capture and intelligent answers also enable teams to diagnose and remediate issues faster while making better-informed release decisions.

Security
DevOps encourages continuous security based on testing, monitoring, authorization, and inventory tracking. Security testing ensures that security is part of the CI/CD process, covering the full software development lifecycle (SDLC). Also known as DevSecOps, these practices add application security testing into DevOps processes and pipelines from their inception to mitigate vulnerabilities.

Cross-team collaboration
Breaking down team and data silos is paramount to ensuring unification across the DevOps pipeline. Effective DevOps execution means establishing a single source of truth—aggregating data from many sources into one collective location, creating a continuous, cross-team feedback loop.

dynatrace

What is platform engineering?

Platform engineering is a software development and operations discipline that helps software teams build self-service IT capabilities for cloud-native environments. A platform encompasses a set of tools, services, and infrastructure that enables developers to build, test, and deploy software applications.

Platform engineering has emerged as a popular and effective approach for improving developer efficiency and satisfaction. The discipline of platform engineering arose primarily in response to the growing complexity of environments and the subsequent need for organization-wide scalability, self-service, and automation capabilities.

Overall, the central purpose of platform engineering is to support DevOps teams with tools, platforms, and workflows for effective software development. To achieve this, platform engineering often incorporates multiple tools and approaches tailored to the organization's unique needs.

The consistency and flexibility inherent to platform engineering result in scalable solutions that reduce friction between teams. Recognizing this, organizations are adopting microservices architectures and shifting from traditional product teams to teams that apply more widely distributed platform engineering principles. Because platform engineering complements DevOps so well, the two disciplines are tightly interconnected.

Internal developer platforms: The platform in platform engineering

Internal developer platforms (IDPs) are the platform in platform engineering. An IDP is an integrated set of tools, services, and infrastructure that platform engineering teams use to streamline, automate, and enhance the software development process. These platforms are tailored specifically to the needs, demands, and goals of an organization. Platform engineering teams design strong and effective IDPs to help them incorporate and manage the following functions within their organization:

Self-service templates
Platform engineering uses IDPs that promote DevOps best practices through self-service templates that follow modern IT architecture, such as microservices.

Application containerization
Platform engineering teams often use IDPs to help streamline appplication containerization, a process that bundles separate application components for greater efficiency. In the context of application containerization, IDPs enable increased consistency and easier management.

Infrastructure as Code (IaC)
With an Infrastructure as Code approach, platform engineers use IDPs to provision and manage infrastructure through code and automation instead of manual configuration. By reducing manual effort, teams can save time and resources.

Security
IDPs also help platform engineers take a DevSecOps approach to security, wherein teams bake security measures into platforms from their inception, providing security protection across the entire organization.

Compliance
To establish strong governance practices, platform engineers use centralized IDPs to ensure their platforms and the actions they enable comply with organizational guidelines and frameworks. This offers consistency and efficiency in meeting compliance standards.

Automation
Platform teams embrace automation as a solution for streamlining key processes, incorporating it into IDPs to manage tasks such as infrastructure scaling, configurations, continuous testing and validation, and pipeline management.

The observability challenge of IDPs and platform engineering

To achieve optimal platform performance and minimize issues, platform engineering teams rely on observability data to gain a better understanding of infrastructure and its performance, metrics, and potential issues or vulnerabilities.

Although an IDP may include some monitoring and logging capabilities, deployment metrics, and environment statistics, it is not a primary observability tool. As such, IDPs can be prone to data silos, lengthy incident management processes, and other complications when not adequately monitored. Observability fills this gap by offering teams the insights needed to properly manage and fine-tune their IDPs.

To learn more, refer to the Observability guide to Platform Engineering.

Overall, the central purpose of platform engineering is to support DevOps teams with tools, platforms, and workflows for effective software development.

CHAPTER TWO

Four ways DevOps and platform engineering complement one another

Cross-functional emphasis

DevOps methodology encourages collaboration between development and operations teams. Platform engineering offers DevOps teams a centralized platform for their tools and workflows to aid this collaboration.

Like DevOps, successful platform engineering requires collaboration between development, security, and operations teams.

Observability

Both disciplines use observability to understand system performance. DevOps employs it to monitor the software's operation, identify issues, and enhance performance. Platform engineering relies on observability to maintain and optimize supporting infrastructure.

The shared emphasis improves team collaboration, enabling proactive issue resolution. Through observability, DevOps and platform engineering collaborate to ensure optimal software and infrastructure performance for a seamless user experience.

Automation synergy

Automation is also a central tenet of DevOps and platform engineering. Both disciplines rely on automation to continuously deliver value with minimal human intervention.

While DevOps teams use automation to consistently build, test, and deliver software, platform engineers use it to streamline the deployment and management of software and infrastructure by investing in tools to provision, configure, and scale resources, enabling more agile and efficient operations.

Feedback loop

Both stress continuous feedback for process enhancement. DevOps utilizes feedback to foster collaboration between development and operations, gaining insights into software performance and deployment. Similarly, platform engineering relies on feedback to ensure infrastructure stability.

This emphasis on feedback enhances communication and collaboration, driving both teams to automate tasks, support CI/CD practices, and create an efficient software delivery environment within a stable infrastructure. This collaborative effort accelerates software delivery and enhances the user experience.

CHAPTER THREE

Four benefits of DevOps and platform engineering

DevOps benefits

Accelerated development cycles
Using DevOps practices, such as CI/CD pipelines, automation, and feedback loops, organizations can accelerate development, testing, and deployment cycles. As a result, teams can greatly reduce time to market for new features and products.

Improved collaboration
DevOps promotes collaboration between development and operations teams, fostering better communication and goal alignment. Since DevOps constitutes a cultural shift among cross-functional teams, it embeds collaboration and communication in the framework of DevOps methodologies.

Increased reliability
Automation enables teams to quickly identify and remediate software defects, reducing the likelihood of outages and downtime. Additionally, continuous monitoring gives full visibility into digital environments, which provides teams with real-time insight into application performance and enables quick response times.

Enhanced scalability
DevOps enables teams to easily scale infrastructure and applications, accommodating changing workloads and user demands. Through continuous observability, infrastructure can automatically scale based on specific environmental cues, such as increased or reduced traffic.

Platform engineering benefits

Standardization and consistency
Platform engineering establishes standardized environments and tools through the creation of IDPs, ensuring consistency and predictability in application deployment. Reusable components, centralized libraries, predefined templates, and other aspects of platform engineering ensure that developments are congruous and cohesive.

Efficient resource utilization
Platform engineers design and manage infrastructure for optimal resource use, reducing waste and associated costs. Resource optimization becomes far easier since platform engineers tailor platforms to an organization's unique needs and capacities.

Enhanced developer productivity
IDPs provide developers with self-service tools and predefined configurations, enabling them to focus on coding rather than infrastructure management. With this increased bandwidth, developers gain increased opportunities to innovate and deliver higher-quality software more quickly.

Scaling DevOps
Platform engineering excels in scaling DevOps by providing a foundation that supports the seamless expansion of DevOps practices among diverse technologies and teams. Through standardized infrastructure, automation, and holistic management, platform engineering ensures that the principles of collaboration, automation, and continuous improvement integral to DevOps can be efficiently extended and sustained as organizations grow and evolve.

dynatrace

Four challenges of DevOps and platform engineering

DevOps challenges

Cultural resistance
DevOps often requires a cultural shift, and resistance to change from traditional siloed structures can be a significant challenge. The key is gaining executive buy-in, which can help ensure overarching business goals and resources align more closely with embracing the new culture. Teams can garner executive buy-in by demonstrating the cost optimization, efficiency, and collaboration inherent to DevOps.

Security
If not properly managed, rapid deployment can introduce new vulnerabilities that necessitate a strong focus on security within the DevOps process. By expanding DevOps to a DevSecOps approach, security becomes a key component of the SDLC from the outset.

Siloed automation
As DevOps matures, teams often build automation in silos, neglecting collaboration and a holistic approach. This fragmented development leads to technical debt, accumulating overhead and complexity as each team's isolated automation efforts may lack cohesion, creating a system that is harder to maintain, integrate, and evolve.

Complexity
Managing a complex DevOps toolchain and integrating various tools can be time-consuming, expensive, and challenging, requiring expertise and careful configuration. Organizations can overcome toolchain complexity by investing in platforms rather than point solutions and embracing a single source of truth for data collection and analysis.

Platform engineering challenges

Performance optimization
Achieving and maintaining optimal scalability and performance is a constant challenge. As user demands fluctuate, platform engineers must design and implement solutions that scale efficiently, manage increased load, and ensure optimal resource utilization and cost-effectiveness.

Security
Platform engineering involves navigating the intricate landscape of security and compliance requirements. Platform engineers must establish robust security protocols, implement best practices for data protection, and ensure adherence to industry regulations, all while balancing the need for security with the desire for a frictionless user experience.

Alignment with development teams
Teams must strike a delicate balance between ensuring that an IDP meets the needs of development teams and simultaneously maintains platform standards. Transparency, well-defined service-level objectives (SLOs) and key performance indicators (KPIs), frequent communication, and a unified understanding of overarching business goals are crucial to navigating such issues.

Collaboration and communication
Effective collaboration between cross-functional teams is a persistent challenge in platform engineering. Overcoming silos, ensuring clear communication, and fostering a collaborative culture are essential for aligning the diverse skills and perspectives within the organization, promoting efficiency and avoiding misunderstandings that can lead to bottlenecks or errors in platform development and maintenance.

CHAPTER FIVE

Four ways platform engineering scales DevOps in a cloud-native world

1. Goals

Platform engineering focuses on scaling DevOps across an organization by providing IDPs equipped with standardized self-service capabilities tailored to the needs, requirements, and skill sets within the IT organization.

2. Areas of responsibility

Platform engineering teams are responsible for providing “DevOps as a self-service,” seeking to address the scale concerns of the “you build it, you own it” mentality. This includes understanding the organization-specific use cases, tooling, and underlying infrastructure needed for engineering teams to build and ship their applications.

3. End users and user experience

Platform engineering amplifies the impact of DevOps on end users by creating a resilient and scalable infrastructure that supports continuous improvement. Through meticulous management of the underlying platform, infrastructure, application build and rollout processes, platform engineering ensures consistent and reliable service delivery, optimizing the end-user experience with enhanced system reliability, reduced downtimes, and improved performance.

4. Key metrics and objectives

While faster software delivery, increased frequency of deployments, and reduced software defects and downtime are the key DevOps objectives, platform engineering’s north star is end-user satification. However, this end-user satisfaction is achieved by ensuring ‘platform success,’ which is measured through metrics such as platform downtime, resource utilization, and service provision time. In addition, ensuring a positive developer experience (DevEx) is critical and is measured through metrics such as developer satisfaction, productivity, and efficiency.

Platform engineering ensures consistent and reliable service delivery, optimizing the end-user experience with enhanced system reliability, reduced downtimes, and improved performance.

Four evolving practices in DevOps and platform engineering

Large language models (LLMs)
As breakthroughs in generative AI emerge, the role of LLMs will become increasingly essential to the efficacy of DevOps and platform engineering practices. For one, LLMs enable teams to make intuitive natural language queries regarding infrastructure (such as health, inventory, configurations, logs, and performance metrics).

Organizations can expect LLMs to improve deployments, testing, code quality, delivery pipelines, and security.

GitOps
To properly manage infrastructure amid rising demands, organizations are beginning to rely more heavily on GitOps. The ability to unify all infrastructure and application assets within a single Git repository and automate their definitions streamlines DevOps and platform engineering practices alike.

A single source of truth such as Git breaks down silos, promotes collaboration, and boosts efficiency across teams. Furthermore, by automating infrastructure definitions for DevOps and platform engineering processes, GitOps helps teams accomplish their SLOs.

Everything as code (EaC)
Building off of infrastructure as code (IaC), everything as code (EaC) aims to apply code-based definitions to virtually all DevOps and platform processes. EaC is made up of several “as code” approaches, such as configuration as code, deployment as code, observability as code, and remediation as code.

Using such approaches, teams can efficiently manage and scale platforms, operations, and resources. Additionally, teams can make processes more streamlined and repeatable with EaC.

Developer experience (DevEx)
DevEx refers to the quality of the elements that enable and enhance developers’ ability to build and deliver software. Such elements may consist of the tools, policies, processes, technology, and culture that affect developers’ workflows.

Since these components have a substantial influence on the software quality that developers produce, a positive developer experience is key to an organizations’ success, and strong, effective IDPs play a pivotal role in this experience. A greater investment in DevEx will provide organizations new avenues for innovation within DevOps and platform engineering.

CONCLUSION

Unified observability and security for DevOps and platform engineering

For organizations looking to transform their ability to deliver value, the implementation of DevOps and platform engineering practices are non-negotiable. Both disciplines are not just beneficial initiatives, they are imperatives that have become increasingly central to the productivity and success of organizations dealing with cloud-native technology.

Given the evidence that the two practices are complementary rather than opposing forces, platform engineering can be thought of as a way for an organization to mature its overall DevOps goals. Platform engineering hones DevOps practices through provisioning centralized self-service solutions for engineering teams at scale. These solutions come in the form of IDPs and are specifically designed to address the needs and capabilities of an organization, allowing teams to better align and deliver higher quality software faster amidst a potentially complex infrastructure.

In practice, platform engineering helps mature overall DevOps practices by centralizing development tooling, CI/CD toolchains, processes, and lifecycle orchestration solutions into an IDP that development teams can use for greater efficiency. Platform engineering teams implement DevOps practices to create and maintain these platforms, which requires that they understand the use cases for which self-service capabilities teams within their organization need.

Therefore, the debate should not concern the efficacy of DevOps over platform engineering, but rather how teams can best use the two in tandem to deliver the greatest value.

How Dynatrace can help

The Dynatrace unified observability and security platform provides real-time answers and AI-driven automation that integrates telemetry from across multicloud environments and optimizes the output of IDPs, eliminating silos and revealing key end-to-end insights.

With a unified observability and security platform such as Dynatrace, teams can mature their DevOps processes and implement platform engineering at scale. This integrated observability approach gives DevOps and platform engineering teams access to the same reliable information, eliminating blind spots and bolstering automation, so teams can deliver better software faster.

dynatrace

Automatic and intelligent observability for hybrid multiclouds

We hope this ebook has inspired you to take the next step in your DevOps and platform engineering journey. Dynatrace is committed to providing enterprises the data and intelligence they need to be successful with their transformation initiatives, no matter how complex.

Learn more

If you are ready to learn more, please visit www.dynatrace.com/platform for assets, resources, and a **free 15-day trial.**

02.07.24 10026_EBK_cs