

Elevating DevOps: How Platform Engineering Mitigates DevOps Challenges in Modern Software Development

Sagar Chhabra

Department of Computer Science & Engineering
IET, SAGE University
Indore, India
sagarchhabra02@gmail.com

Prof. Kundan Kumar Mishra

Department of Computer Science & Engineering
IET, SAGE University
Indore, India
kundankumarmishra@gmail.com

Abstract—DevOps was introduced within organizations to bridge the traditional divide between development and operations. By implementing DevOps practices, companies aim to dismantle silos between developers and operations teams through a continuous loop that includes continuous integration, deployment, monitoring, operations, and implementing everything through continuous collaboration. While a DevOps mindset resolved many issues in software development, there are some challenges as well in implementing a DevOps culture within a team. This paper highlights the challenges faced by teams in the implementation of DevOps practices and how some of those challenges can be mitigated through the implementation of Platform Engineering.

Keywords—DevOps, Platform Engineering, Operations, Challenges, Mindset

I. INTRODUCTION

Development and Operations are two sides of the same coin. Both are key parts of building robust software and working on improving the end user's experience. Over the years, organizations have adopted many process optimizations in their software development, but the focus has been mostly on the development side [1]. Although both development and operations share similarities in their end goal, they both differ in their responsibilities. This gap between them can cause many issues in the software development lifecycle if the team does not have the right mindset to achieve the common goal of providing robust software to end users.

DevOps is the mindset that splits the barrier between developers and operations personnel [2]. To overcome such a split that is predominant in many organizations today, organizational changes, cultural changes, and technical frameworks are required [3]. Platform Engineering shares the same goal but it focuses more on the technical aspects of building and maintaining the infrastructure and tools to support software development throughout its lifecycle.

II. DEVOPS

Software development involves multiple stages — planning, integration, deployment, monitoring, and testing. The DevOps mindset is applied to each stage for faster delivery of software applications.

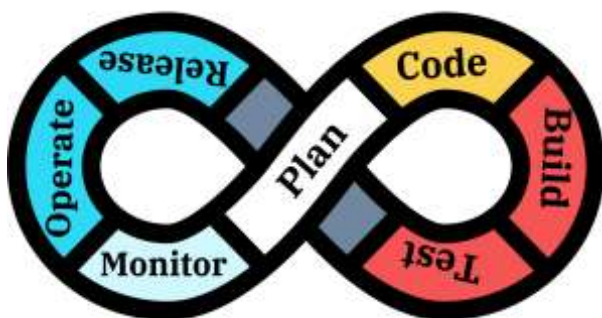


Fig. 1. The DevOps infinity loop

Continuous Planning: As market conditions are changing rapidly, business plans need to adjust quickly to market conditions [1]. DevOps practices promote active collaboration by maintaining a backlog of items that are updated and modified actively for development and testing teams. Work items are continuously tracked by business as well as development teams for active collaboration and managing their work as per updates in the plan. Teams follow agile methodologies for flexibility, adaptability, and a customer-centric approach.

Continuous Integration: This stage refers to building the code early to find issues in the initial stages of code development. This is achieved by setting up build pipelines which are triggered based on code commits in a particular branch and repository. The code is built as soon as the developers push their code and the build informs the developers about any issues during the development phase. This is a repeatable process throughout development [1].

Continuous Deployment: The DevOps team sets up pipelines for releasing the code to the respective environments. The code is deployed to the environment as early as possible and any kind of infrastructure issues are found early. The software team maintains a separate development environment which is used for code deployment during the development phase. Dev environments are useful in identifying any kind of infrastructure or integration issue in the initial phases.

Continuous Monitoring: Monitoring is very crucial in software development to understand the current status of the environment and find issues as early as possible. DevOps uses several tools to continuously monitor environments and find issues sometimes even before they become noticeable to end users. Proper alerting mechanisms are also set up so that the stakeholders are aware of the issue faster.

Continuous Testing: Testing can also be integrated into pipelines so that it can be done with the development itself. Nowadays, development and testing teams also work together so that they can find issues as they are getting developed. Continuous testing not only moves the testing to early stages but also allows us to do testing in production environments [1].

III. PLATFORM ENGINEERING

Platform Engineering is built on top of the DevOps mindset and shares the common goal of building a robust software application and delivering it faster to end users. Platform Engineering focuses more on the technical side of software development. A Platform Engineer is responsible for designing, implementing, and maintaining the infrastructure and tooling necessary to support DevOps practices within an organization. Platform Engineers can be shortly defined as “Software Engineers for the Development teams.” Platform Engineering operates on the principle that developers should have the capability to both develop code and deploy it into production environments. Platform engineers should be the ones facilitating this process, empowering developers to manage their deployments rather than handling every aspect themselves.

Platforms can eliminate the issues that might occur during the development process and they provide highly optimized developer experiences and simplified operations. Platforms include tools that support developers in the following ways [4]:

- Help developers be self-sufficient.
- Assist with common day-to-day tasks.
- Encapsulate common patterns and practices into reusable building blocks.
- Simplify operations by managing underlying infrastructure and tools.
- Provide early advice and feedback on problems with the application or infrastructure.

Platform Engineering teams utilize the Internal Developer Platform (IDP) to build golden paths and enable developer self-service. The platform team treats their platform as a product and builds it based on user research which includes taking feedback from development teams who are their end users. They also maintain it and continuously improve it to make the process easier for development teams [5].

An Internal Developer Platform should be built to cover five core components, which are as follows [6]:

- 1) Application Configuration Management: Configurations of applications should be managed in a dynamic, scalable, and reliable way.
- 2) Deployment Management: Deployment pipelines should be built for the continuous delivery of applications.
- 3) Infrastructure Orchestration: Infrastructure should be orchestrated using infrastructure-as-code tools and containerization technologies dynamically and intelligently.
- 4) Role-Based Access Control: Access should be managed in applications and environments to enhance security.
- 5) Environment Management: Developers should be able to provision environments based on requirements.

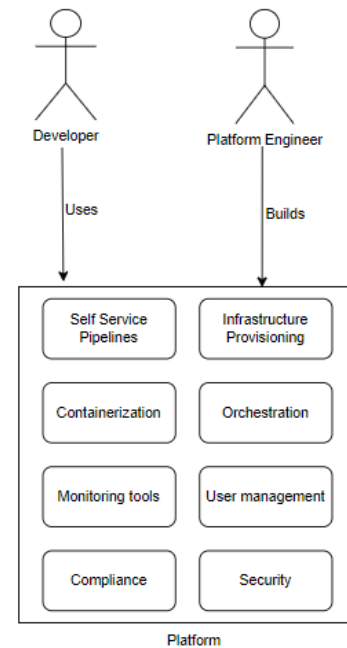


Fig. 2. Platform Engineering setup

IV. CHALLENGES IN DEVOPS IMPLEMENTATION

DevOps promises streamlined collaboration and accelerated development cycles, but its implementation is not without obstacles. The complexities faced in the implementation of DevOps pose challenges for organizations aspiring to adopt DevOps successfully. In the discussion below, we explore key hurdles hindering the seamless integration of development and operations processes.

We conducted a survey of DevOps engineers from multiple organizations and projects in our previous work [12], through which we came across multiple challenges faced in the implementation of DevOps practices in actual working environments. Based on our key findings from the survey and work done by other authors, we found common challenges faced by projects in bridging the development and operations gap.

A. Communication Challenges

One major challenge faced in the implementation of DevOps is proper communication among teams, both internally and with other teams. DevOps is a culture that bridges the gap between development and operations. It promotes continuous collaboration, but it is very challenging to implement this in the real world. DevOps practices can be easily introduced in a team, but it is required that every team member adopt these practices and narrow the communication gap within teams internally and externally. Etgar [7] noted that if there is no clear communication within teams, conflicts can occur, which leads to misunderstanding, incorrect strategies, and mutual feelings of frustration.

Khan et al. [8] also highlighted some factors that can challenge communication, as follows:

- **Geographical distance:** If there is a high geographical distance between team members, it will require more effort

to visit one another. Although geographical distance is slowly becoming irrelevant with the increase in technology and remote-work culture, it still provides significant benefits when team members can meet with one another.

- **Socio-cultural distance:** This involves national culture, organizational background, policies, and moral norms [9]. This also presents a challenge in implementing DevOps in a team, as people from different socio-cultural backgrounds have different work cultures and habits.
- **Temporal distance:** This refers to the time difference between different time zones of different team members. Having a global team provides certain benefits to the product, as people with different mindsets and cultures connect and contribute new ideas. However, it also introduces challenges, because temporal distance makes it harder for teams to proactively collaborate. Usually, team members have to wait for their dependent items, as working hours are typically different.

B. Undefined and Varied Role Definition

The term DevOps is widely misunderstood in organizations. Organizations and teams have modified it based on their use case, which sometimes provides benefits in the short term, but usually at the long-term cost of the benefits provided by DevOps. Most organizations divide responsibilities as follows:

- The one who develops — software developer.
- The one who tests — software tester / QA.
- The one who sets up infrastructure, pipelines, permissions, deployments, configurations, and connections with other teams — DevOps.

Organizations leave varied levels of responsibility with DevOps. Because of this, there is no clear definition of DevOps, which is one of the biggest challenges in implementing DevOps in organizations.

When DevOps personnel change organizations, they are also not aware of the practices of the new team. Usually, DevOps personnel have to adopt the practices of new teams rather than implementing correct DevOps practices from scratch. This would have been easier if there were a standard definition for DevOps, but in reality, DevOps is a set of practices that vary across organizations.

C. Tool Abundance

Being a DevOps engineer is a challenging responsibility. New tools are launched and adopted by organizations constantly. On the other hand, some organizations have been using standard tools for many years. This creates a market for DevOps where there are many tools serving the same purpose. Every team has a separate set of tools, and all the tools come with their own set of practices, which sometimes makes it difficult to implement DevOps practices consistently.

Here are some DevOps tools for each stage of the software development lifecycle [10]:

- **Agile Project Management:** Jira, Confluence, Azure Boards, Zoho, Trello, GitHub Project Management.
- **Collaboration:** Slack, Teams, Zoom.
- **Source Control:** GitHub, Azure Repos, Bitbucket, GitLab.
- **Code Integration:** Jenkins, CircleCI, TeamCity, TravisCI, GitHub Actions, Bamboo, Azure Pipelines, AWS CodePipeline.
- **Code Deployment:** Jenkins, AWS CodeDeploy, Octopus Deploy, GitLab, Azure Pipelines.
- **Testing:** Selenium, Tosca, JMeter.
- **Monitoring:** Prometheus, Grafana, Datadog, Splunk, Nagios, Zabbix.
- **Infrastructure:** Azure, AWS, GCP, Oracle Cloud, Alibaba.
- **Infrastructure as Code (IaC):** Terraform, Pulumi, Azure Bicep, AWS CloudFormation.
- **Automation Scripting:** Python, Bash, PowerShell.
- **Containerization:** Docker, Containerd, Podman.
- **Orchestration:** Kubernetes, Docker Swarm, Nomad, Rancher, Mesos.



Fig. 3. DevOps tools

The abundance of tools in the market is necessary, as each tool has a specific purpose and teams choose tools based on their requirements. This also creates challenges for DevOps personnel, as there is a learning curve involved with each new tool. All areas of software development require learning new tools and technologies, but it is more crucial for DevOps because of the sheer abundance of tools.

V. MITIGATING DEVOPS CHALLENGES WITH PLATFORM ENGINEERING

A. Decoupling Dependency on DevOps

There is a difference between the DevOps mindset and DevOps personnel. DevOps personnel are the ones who implement the DevOps culture in the team. Due to the abundance of tools for implementing various stages of software deployment, it is very hard for DevOps personnel to instill the DevOps mindset among teams. Teams usually become dependent on DevOps and rely on DevOps personnel for all major infrastructure- and deployment-related activities.

Platform Engineering plays a crucial role in decoupling DevOps dependencies by providing a stable and standardized

foundation for development and operations teams. Platform Engineering works on the self-service principle. Platform Engineers don't do the work for development teams; instead, they build automation and the required tools so that teams can accomplish their tasks using those tools.

Platform Engineers are “Software Engineers for Software Engineers.” They are just like any other software development team, but their end users are other software engineers. They build tools that developers can use to accomplish their infrastructure- and deployment-related tasks. Platform Engineering decouples the dependency on DevOps in several ways:

Automation of Deployments: This includes automating repetitive tasks and building a process that developers can use for the deployment of their applications. Automation reduces the need for constant communication and coordination between teams for routine activities.

Platform Engineers set up automation as per requirements given by development teams, but they are not required during the deployment process itself. They provide the automation with an approval mechanism, and development teams are responsible for deploying their applications to the respective environments. This promotes faster deployments, as the developers building the application are also responsible for managing the deployments, so they are not dependent on any other team.

Standardization of Workflow: Consistency is very important in setting up software infrastructure. Teams manage multiple environments with separate rules for accessing them. Maintaining standards for infrastructure and deployments is very important for decoupling the dependency between development and operations. Platform Engineers standardize processes that can be easily followed by application teams, so those teams are not dependent on any other team to support them with routine tasks. There are several areas in which processes are standardized:

- **Consistency in Environments:** Platform Engineers use infrastructure-as-code (IaC) tools like Terraform and Pulumi to deploy infrastructure. The same configuration can be used to deploy multiple environments. Network and security rules can be configured directly in the IaC tool to maintain consistency across environments.
- **Documentation:** Platform Engineers need to maintain proper documentation for the tools and processes they develop, since they are not required to be present when their tools are being used. Platform Engineers create documentation that application teams can refer to while using those tools, which standardizes the workflow and makes application teams self-sufficient.
- **Monitoring and Logging:** One of the major responsibilities of the operations team is monitoring the infrastructure and application for anomalies and errors, taking appropriate action proactively. Platform Engineers set up proper monitoring mechanisms so that development teams can also monitor the performance of their applications in real time. Tools like Prometheus, Grafana,

and Nagios collect required metrics from applications and present them as dashboards, and proper alerting mechanisms can be set up to notify stakeholders early.

This also allows the developers who built the application to monitor it using dashboards built by platform engineers, resulting in a decoupling between development and operations.

Self-Service Platforms: Platform Engineers create self-service platforms that let application teams manage their applications independently. This reduces reliance on the operations team or DevOps personnel for routine tasks.

This decouples the dependency on DevOps, and the tools made by platform engineers are used directly by software engineers.

B. Abstraction of Processes

Technologies have been abstracted from the beginning to ease the development process and create room for new technologies. Some notable abstractions include:

- Assembly language abstracted binary language for instructing a machine to complete a task.
- High-level languages abstracted low-level languages to simplify development.
- Programming frameworks abstract code to simplify building software.

Not everyone needs to understand everything to be able to use technology. A driver does not need to know the mechanics of an engine and brakes to be able to drive a car. Similarly, Platform Engineers abstract infrastructure and deployment mechanisms and expose only the part that developers need, as per their requirements.

Public cloud platforms have been abstracting infrastructure by building scalable and secure platforms used by development teams. Cloud providers offer platforms in the form of Platform as a Service (PaaS). PaaS services provide different levels of abstraction for accomplishing a particular task. Here are some PaaS examples:

- 1) Azure App Service, AWS Elastic Beanstalk, and GCP App Engine are platform services for deploying applications; the underlying infrastructure and networking are managed by the cloud platform.
- 2) Azure SQL Database, AWS RDS, and GCP Cloud SQL are used to manage databases in the cloud, providing high scalability and authentication mechanisms.
- 3) Azure Storage Accounts, AWS S3, and GCP Cloud Storage are used for storage, providing versioning, security, and connectivity for stored data.

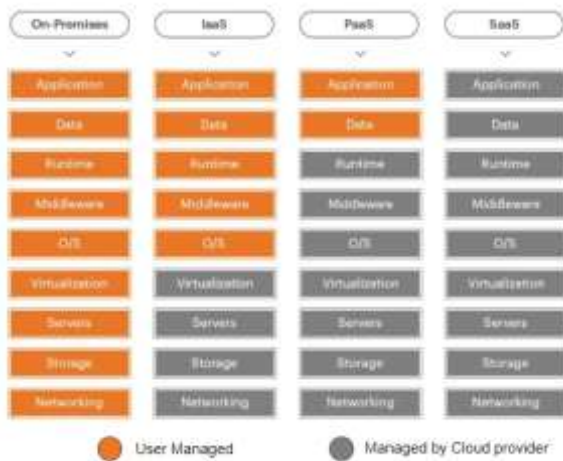


Fig. 4. Responsibility split across on-premises, IaaS, PaaS, and SaaS

Just as public cloud platforms have built platform services to abstract underlying infrastructure and let developers easily use their services, teams can rely on their own platform engineers to further abstract services and let developers manage their deployments and applications more easily.

The level of abstraction is an architectural decision that depends on individual team requirements. The level of abstraction is inversely proportional to the level of control a developer has over the application and infrastructure. Platforms are made flexible based on project requirements.

Organizations can also have organization-level platform engineering teams that abstract common practices used across the organization.

Building a platform is also a multi-level activity. A platform can be built on top of another platform, creating one more level of abstraction. Platforms are built for individual use cases, and the right level of abstraction is decided based on the architecture of the application.

VI. POSSIBLE FUTURE OF PLATFORM ENGINEERING

The future of Platform Engineering involves new layers of abstraction on top of the current ones. Platform Engineers will build new platforms by identifying current issues and will make processes easier for development teams. The dependency on DevOps will be reduced, and teams will work more independently. Developers will have more control over the infrastructure and deployment of their applications, while things that are irrelevant to their application will be abstracted away.

The responsibilities of DevOps personnel will change. There will be some key shifts in operations as teams start implementing a platform-engineering approach on top of the DevOps mindset:

- Developers will have more responsibility over their applications; they will build, deploy, and monitor their applications themselves.

- DevOps personnel will work in areas that are not automated, either intentionally or because automation has not yet been considered.
- Large-scale projects will have their own platform engineering team, working to make the process more efficient for development teams.
- Small-scale projects will share a platform engineering team, with a central team managing platforms for multiple small projects.

Platform Engineering is also not a replacement for DevOps in any way. It simply creates new layers of abstraction that simplify the process for development teams, and it supports DevOps processes rather than replacing them. Platform Engineering is simply the next progression in delivering modern application infrastructure [11].

VII. CONCLUSION

Concepts of DevOps and Platform Engineering are explained in this paper, along with the challenges faced in implementing DevOps. Platform Engineering is not a new technology; rather, it is a set of tools and techniques for facilitating infrastructure and deployment of applications, something that has been done for a long time, but the concept is continuously evolving and shaping into a mature set of methodologies. Therefore, Platform Engineering is not a replacement for DevOps, but simply an evolution of the same philosophy.

References

- [1] M. Virmani, "Understanding DevOps & bridging the gap from continuous integration to continuous delivery," Development Manager, IBM Rational, 2015.
- [2] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, USA: Pearson Education, 2010, p. 512.
- [3] J. Wettinger, U. Breitenbücher, and F. Leymann, "DevOpsLang — bridging the gap between development and operations," in Service-Oriented and Cloud Computing, ser. LNCS, M. Villari, W. Zimmermann, and K.-K. Lau, Eds. Springer, 2014, vol. 8745, pp. 108–122.
- [4] Microsoft Learn, "What is platform engineering?," para. 3, learn.microsoft.com, 13 Nov. 2023. [Online]. Available: <https://learn.microsoft.com/en-us/platform-engineering/what-is-platform-engineering>
- [5] Internal Developer Platform, "What is an Internal Developer Platform (IDP)?," para. 1, internaldeveloperplatform.org, 25 Jan. 2024. [Online]. Available: <https://internaldeveloperplatform.org/what-is-an-internal-developer-platform/>
- [6] Internal Developer Platform, "What is an Internal Developer Platform (IDP)?," para. 1, internaldeveloperplatform.org, 25 Jan. 2024. [Online]. Available: <https://internaldeveloperplatform.org/core-components/>
- [7] M. Etgar, "Effects of administrative control on efficiency of vertical marketing systems," Journal of Marketing Research, vol. 13, no. 1, pp. 12–24, 1976.
- [8] A. A. Khan, S. Basri, and P. Dominic, "Communication risks in GSD during RCM: Results from SLR," in Proc. Int'l Conf. on Computer and Inf. Sciences, Kuala Lumpur, Malaysia, 2014. IEEE, pp. 1–6.

- [9] H. Holmstrom Olsson, E. Ó Conchúir, P. J. Agerfalk, and B. Fitzgerald, “Global software development challenges: a case study on temporal, geographical and socio-cultural distance,” in Proc. Int'l Conf. on Global Software Eng., Florianopolis, Brazil, 2006. IEEE, pp. 3–11.
- [10] Atlassian, “DevOps tools,” atlassian.com. [Online]. Available: <https://www.atlassian.com/devops/devops-tools>
- [11] Pluralsight, “Platform engineering: a DevOps evolution, not a replacement,” para. 2, pluralsight.com, 18 April 2023. [Online]. Available: <https://www.pluralsight.com/blog/teams/platform-engineering-devops-difference-explained>
- [12] S. Chhabra, K. K. Mishra, and S. Patel, “A survey of DevOps practices and implementation,” IJRAR — International Journal of Research and Analytical Reviews, E-ISSN 2348-1269, P-ISSN 2349-5138, vol. 11, no. 1, pp. 503–509, Feb. 2024. Available: <http://www.ijrar.org/IJRAR24A1832.pdf>