appvia

DevOps Initiatives Will Fail Without Developer Self-Service

Content Page

DevOps Initiatives Will Fail Without Developer Self-Service

DevOps is a constantly evolving and advancing field. While the goal of delivering high-quality applications and services at high velocity has remained the same, how that's done is always in flux. With **nearly 80 percent of organizations** still in the middle phase of their DevOps journey, the tech industry has seen a mass increase in products and processes to help organizations implement a DevOps culture.

This DevOps movement has driven software vendors to release new tools and find new ways of delivering software in the shortest possible time with the ultimate goal of complete seamless continuous integration and continuous delivery (CI/CD).

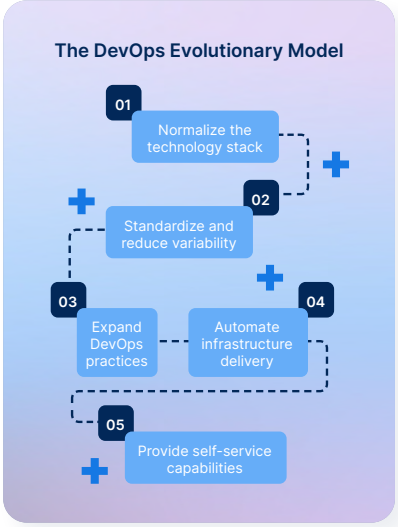
In their **2023 State of DevOps Report,** Puppet found that automation alone doesn't make you a high-performing DevOps team. Around two-thirds of middle-phase organizations have high automation, but cite organizational siloed code deployments as holding them back. This means developers still rely on other team members for many tasks, like provisioning cloud environments or accessing computing resources for testing. Each of these delays decreases the speed at which code can be written and deployed, bringing you further away from the continuous goal.

> **"the percentage of people reporting the use of public cloud, including multiple clouds, is now 76 percent, up from 56 percent since last year."**
>
> Google's 2022 State of DevOps Report

This migration to the cloud means that the tools developers and DevOps engineers use are more distributed and interconnected than ever before. This means a broader disparity of resources, programs, and processes, which can lead to further delays and bottlenecks when managing infrastructure and deploying code.

As DevOps has adapted to this fast-paced, interconnected environment, one thing has become obvious: Developer self-service is essential, with 93 percent of respondents to the **Puppet State of DevOps 2023** Report declaring

“that platform engineering adoption is a step in the right direction.”



Developers must be able to complete DevOps tasks independently to achieve the speed, accuracy, and security demanded by the market. Empowering, guiding, and protecting developers is necessary for success, and this whitepaper explains why.

In this paper, we’ll thoroughly explain what self-service is and examine why it is vital moving forward. We’ll also discuss how you can ensure proper security with self-service, even in full cloud environments. And finally, we’ll discuss how developer self-service will likely evolve and how Appvia can help ensure you’re set up for success.

DevOps By Developers

The objective of DevOps is to integrate both development and operations into a single culture. Siloed DevOps teams don’t fully promote this culture, so the industry has turned to self-service. **60 percent of developers** acknowledged self-service DevOps platforms as a key enabler to faster code releases. Developer self-service involves creating automated and reliable processes that allow developers to accomplish traditional DevOps tasks without involving DevOps engineers. Essentially, we want to shift the ability of DevOps to developers — but without the burden of DevOps knowledge.

In an ideal world, developers can quickly spin up infrastructure for development and testing. Environments can be created, controlled, and destroyed without needing to understand the complexity of how those tasks were automated. Everything a developer needs is available to them with little outside help, only focusing energy on the support for more complex tasks. In this perfect world, the developer experience is smooth and frictionless.

Enabling developers to perform necessary, complex DevOps tasks without requiring the related DevOps knowledge allows them to perform their necessary tests with built-in security, scalability, and flexibility. In turn, this speeds up application development and deployment, which is the goal of developer self-service.

What is Self-Service?

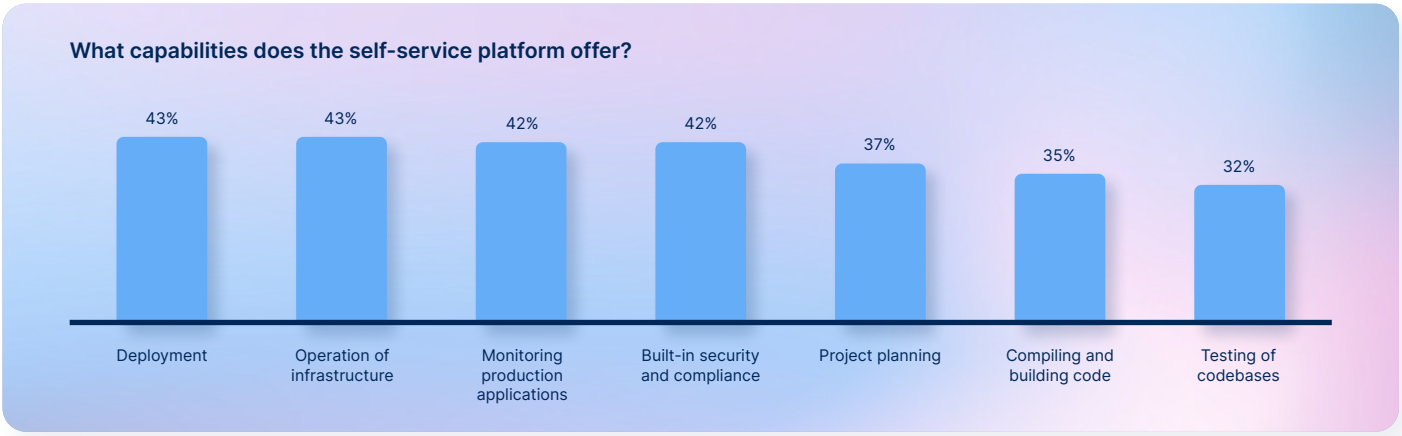
The guiding principle behind developer self-service is that everything is more efficient if developers do not need direct help from operations to get things done. The goal is to allow developers seamless development and deployment without outside interference, with the understanding that developers will know what they need when needed. This is done by giving developers tools and processes that guide them through the setup process, allowing them to perform complex tasks autonomously. This can be anything from allocating server space, access to GPU/CPU, automated deployment, and anything else required for developers to function at a high level.

The concept is simple enough, but in practice, developers need a safe and reliable environment in which they can operate in a self-service manner. Considerations need to be made for security and code quality, as well as the provisioning of resources. As such, developers need to be limited in the scope of their power and ability, ensuring they can do what they need to do but without adding undue risk.

This requires that the platform team curate an environment that guides developers through self-service. Guardrails must be in place to ensure that the developers can't do potentially risky activities and that their resources are limited and controlled. This takes substantial infrastructure and technology and must be constantly maintained. But the result is a system in which the developers are free to build and deploy with little concern for standards.

Why Self-Service?

The Puppet State of DevOps 2023 Report highlighted several business benefits from embracing developer self-service, including "improvement in system reliability to greater productivity/efficiency to better workflow standards." By making things a commodity, we can make the developer experience as simple and efficient as possible. Developers with the information and support needed to make informed decisions can facilitate a smoother deployment. This saves time on deployment and testing and builds scalability into the process.



The **same report** notes that, when asked about speed increases due to the introduction of developer self-service,

> **“68%**
>
> **(of respondents) [were] already experiencing an increase in development velocity.”**

By constructing reliable, efficient processes using automated tools, guardrails can be put in place around teams, allowing them to deliver without feeling the need to perform extra tasks to guarantee a secure outcome.

Developer self-service for DevOps is scalable. Because it relies on the automation of tools and products, it can function on larger, more complex projects without additional team resources or human interference, which allows for this scalability. We can see this exemplified in the historical changes to DevOps that we've already outlined.

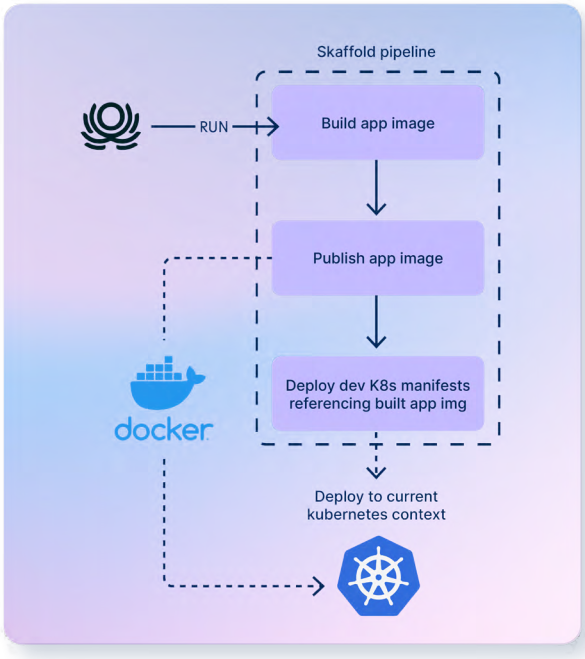
Take containerization, for example. A containerized approach simplifies the deployment through tooling. It allows the application to be deployed in a secure and repeatable way, reducing the overhead for deployment. Before containers, the whole operating system would need to be managed, and the dependencies for the application would need to be constantly updated. With containers, developers can now be in charge of building and distributing their software from CI to development environments.

In addition, the infrastructure that the application was deployed to — both the operating system and hardware — had to be considered. This work fell under the domain of the DevOps platform team. As the application grew in size and users, the platform team had to scale up to continue monitoring, troubleshooting, and properly facilitating deployment and upkeep.

Containerization, however, makes developers responsible for this function, giving them a scalable, insular approach to deploying applications. Containers allow developers to build self-contained, deployable applications that are guaranteed to have the proper dependencies and can function independently from the host system.

This insulated approach means that the infrastructure and maintenance work of the platform team is greatly reduced, as everything needed to run the application is bundled together. Each containerized deployment is designed to work independently so that any number can be deployed without needing additional personnel. This shift away from traditional configuration management tools reduces complexity for the platform team as cloud machine images are baked into systems like Kubernetes, putting into place lightweight operating systems that can run everything as containers.

This is just one example of how developer self-service saves time and resources, giving developers complete control over a DevOps operation.



Self-Service Security

At first glance, it may seem that the shift toward developer self-service can reduce application security even as it increases deployment efficiency. However, this isn't the case. **Google's 2022 State of DevOps Report** found that "participants anticipate a lower chance of security breaches, service outages, and performance degradation as companies increase [the] establishment of supply chain security practices." This is because security best practices can be built directly into the self-service tools.

If you have a defined way of working for developers — from application deployment to infrastructure provisioning and orchestration — then placing the right security checks in the process becomes easier. For self-service to work properly for the business, it's essential to put guardrails in place to ensure that developers can get what they need within the constraints of the business security policies.

The secret lies in the tools now available to developers and DevOps engineers. Modern techniques ensure testing and feedback are implemented as early as possible in the development process. In essence, security is not external to the process — it's an intrinsic part. Building a security framework that ensures every step of the process is secure will lead to fewer issues arising after the application is constructed.

Automated tests can be orchestrated to facilitate continuous testing. Every change sets off a series of interrelated tests, ensuring that configuration and application code are secure and those components are secured when assembled. Automation and orchestration are the keys to project security at every point in the life cycle. The Google's 2022 State of DevOps Report states, "[T]he presence of CI in an organization was a predictor of the maturity of its security practices."

The same process can be applied to security-related settings in deployment. By automating the declaration and testing of deployment settings, self-service tools can ensure accurate and consistent enforcement of security policies across the whole infrastructure. This chore fell to the DevOps team in the past, but controlling cloud deployments at scale requires a platform engineering approach to delivering in the cloud. By enabling platform engineering, you can centralize the policies in one place and distribute them across your estate. Security teams support this as "a consistent platform promises a smaller surface area to secure."

This process places security configuration parameters and policies around what is being self-served and layers them over the top of what developers provide to deliver outcomes. This means that security is automatically embedded at deployment time, regardless of the developer's action. Runtime checks are also required, ensuring security when the cloud automation is running. These approaches are layered, helping to provide consistent and reliable security over the long term.

Cloud Environments

GitLab's 2022 DevSecOps Survey found that cloud computing was high on the list in investment areas, coming second only to security. This investment and migration to the cloud provides more self-service tools for developers, especially since the introduction of **serverless.** However, it also brings some challenges. Building cloud applications in the business — especially when you have multiple teams or applications — requires a lot of infrastructure, security investment, time, and knowledge.

Additional storage is likely needed and must be provisioned and installed. The operating software must be chosen, installed, and configured. Networking is another hurdle, as systems and users must all be able to communicate. And all of this must be done securely.

As if that wasn't daunting enough, an isolated cloud account for every team is required, which must be replicated multiple times. If both production and non-production teams share cloud infrastructure, unintended changes to code, settings, or policies could be pushed at any time, jeopardizing the security of the whole application.

Unfortunately, sharing infrastructure is too common in modern DevOps, as cloud resources can be expensive. Creating multiple environments can also duplicate security risks, as any risks present in one environment are likely to be copied to another.

It's also difficult for a platform team to ensure conformity in settings, resources, and security across multiple instances. The current state of DevOps means that you must manually address a long list of cloud deployment concerns, and it's easier to manage fewer shared resources, which encourages sharing infrastructure.

The shift toward self-service options for DevOps has alleviated many issues with manual cloud management. **Google's research** found "that cloud computing enables teams to excel at things like software supply chain security and reliability, which leads to organizational performance". Containerization and the other technologies discussed above have aided the security and administration of deployments. However, most things like resource provisioning and application management still fall under the domain of a DevOps platform team.

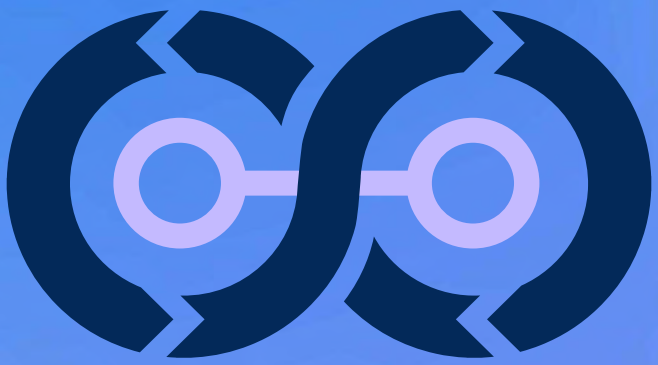
> **"Only 40%**
> **of companies use self-service platforms"**

Only **40 percent of companies** at the lower end of their DevOps journey use self-service platforms. This indicates there is still work to be done for many companies to increase their self-service adoption.

The Future of Self-Service

Platform engineering with developer self-service is becoming the catalyst for efficient application delivery. 94 percent of **Puppet's 2023 State of DevOps Report** respondents agreed that platform engineering and self-service improve their DevOps culture. The most impactful advances in the platform engineering space have been tools that give developers more power and responsibility. This sentiment is echoed in **GitLab's 2022 Developer Survey**, which found that the "group most likely to use a DevOps platform is devs."

It's vital to understand an application's requirements in order to scale successfully and provision cloud resources. Naturally, developers will understand their application more than anyone else, so moving the power into their hands makes sense. A successful self-service approach allows for the automatic provisioning of cloud environments and accounts so that developers can get isolated environments as a by-product of needing an environment for their workload. Teams already automating infrastructure provisioning through a self-service DevOps platform report that the **"use of a DevOps platform was the number one reason for the increased pace of code release."** This is because teams can handle their provisioning requests while spending less time managing and securing resources, as this has been built into the self-service platform.



Containers have been a massive boost to developer-led DevOps, but container management principles rely on methodologies like GitOps and tooling like FluxCD or ArgoCD. However, these methodologies don't scale well with the needed consistency and commoditized approach to the cloud. Careful orchestration by a DevOps team is usually required to ensure balance and efficiency across a scaled container approach. Similarly, infrastructure management has traditionally been removed from the reach of developers. A modern self-service approach must rectify this and shift responsibility to the developers via automated management tools.

One day, all barriers between developer and deployment will be removed. Self-service tools will be the vehicle that gets them there.

Conclusion

Developers are the crucial drivers behind any software product. Empowering them with the necessary tools to execute DevOps tasks swiftly, effectively, and independently marks the future of software delivery.

Scaling poses significant challenges for all companies, notably those dealing with intricate cloud ecosystems. The risks and responsibilities associated with the manual creation and maintenance of cloud environments by DevOps teams are far too great to ignore. The aspiration of a seamlessly integrated and continuously delivered application can only be fulfilled by entrusting developers with meticulously regulated and automated tools.

Several solutions, such as **Appvia Wayfinder**, in the market focus on making this vision a reality. They offer a wide array of self-service tools capable of transforming your development team into a formidable DevOps force that can manage an application from its inception to delivery. These solutions position the tools for cloud operations squarely in your developers' hands, effectively eradicating the obstacles between the developer and delivery. This approach signifies the route to software efficiency.

Regardless of whether you're preparing for cloud migration or have already embarked on the journey, these innovative solutions offer self-service tools to streamline your cloud experience and ensure it's productive and rewarding.