EBOOK

spacelift

Balancing Speed and Control in DevOps

Why platform engineering is the key to safe self-service infrastructure

Introduction

The appeal of centralizing control of your infrastructure management is undeniable: Funneling all your processes through a single team with an in-depth understanding of DevOps best practices ensures control over IaC and pipelines for maximum security and reliability.

However, centralization also means bottlenecks. Waiting times can stretch to weeks as one team with limited resources must deliver everything required to deploy the infrastructure that everyone needs. Sure, everything is secure, reliable, and consistent when your DevOps team members serve as gatekeepers, but watching changes take effect safely and securely before moving on to something else is hardly the best application of their expertise. Fortunately, there is a better way.

In this guide, we look at the shift from DevOps to platform engineering and the central role self-service plays in it. We explore the value of successful self-service systems in organizations and explain how a specialized infrastructure orchestration platform like Spacelift enables self-service. Along the way, we present real-world examples of how it's done and include stories from customers who have safely transformed their developer velocity by building successful self-service workflows.

But first, what is platform engineering — and how does it differ from DevOps?

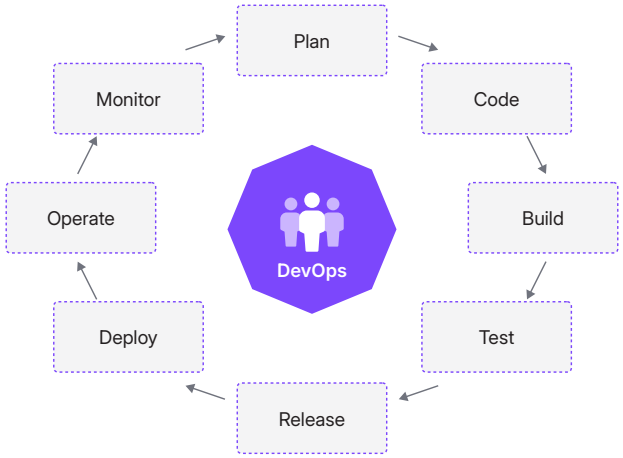
Platform engineering vs. DevOps

DevOps has transformed software development and deployment, shortening development cycles, accelerating deployments, reducing deployment failures, and speeding up recovery from issues. Platform engineering has not superseded DevOps; they are complementary practices. In fact, organizations that have adopted platform engineering generally have a DevOps organization in place already.

DevOps defined

DevOps concepts, methodologies, and best practices are designed to advance the software delivery experience, whereas platform engineering involves creating centralized IDPs that provide essential tools and workflows for developers.

By eliminating the gap between development and operations teams, DevOps shortens the delivery lifecycle, increasing throughput and quality.



Essentially, DevOps is a cultural movement that promotes improvements around developer autonomy, automation, and collaboration. Successful DevOps strategies typically embrace multiple tools and best practices to remove roadblocks from the development process:

- CI/CD pipelines save time and help maintain consistency.
- Repeatable test suites and automated security monitoring ensure issues are detected early in development.
- Developers can easily access live infrastructure to test scalability changes and investigate issues reported by operations teams.

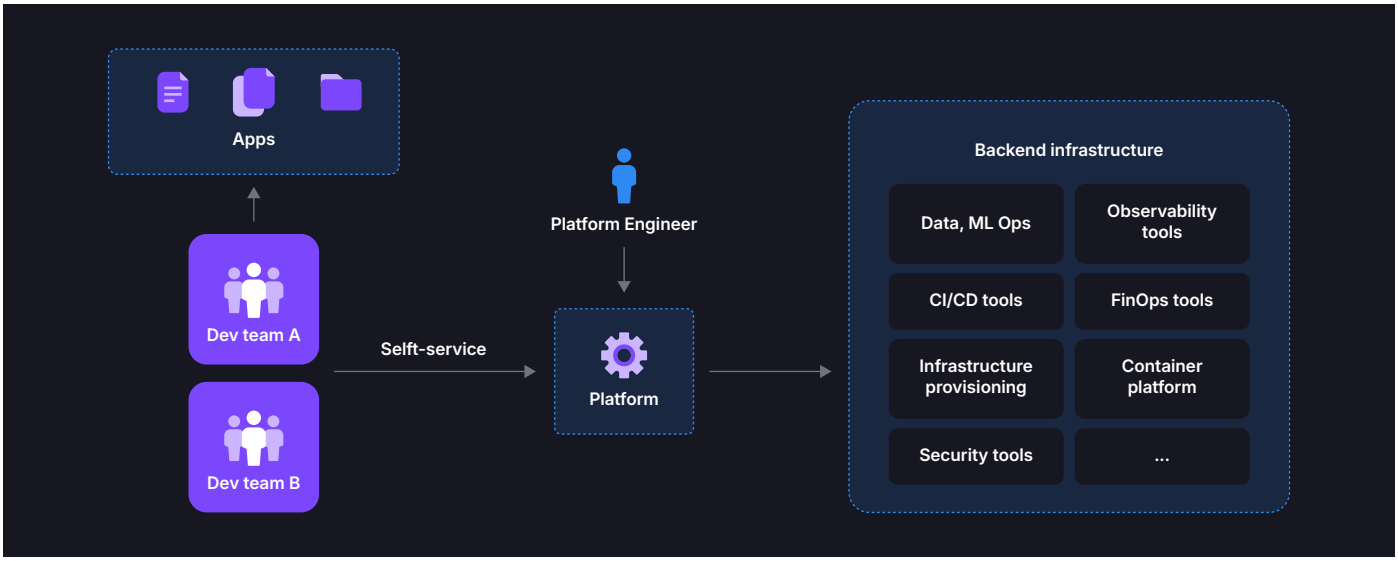
A more responsive development workflow emerges, centering on continuous improvement and close communication. But DevOps teams can struggle to implement these techniques in a way that delivers real value to devs — enter platform engineering.

Platform engineering defined

Platform engineering provides a unified, self-service platform for developers (IDP) by designing and implementing toolchains that streamline software development and delivery. This IDP serves as a bridge between developers and the infrastructure, streamlining complex tasks that would be impractical for individual developers to handle on their own.

Platform engineering doesn't just streamline and standardize the development and deployment process, it also enhances developer velocity, improves reliability and performance, and increases the overall scalability of your infrastructure and applications.

Self-service access is a key plank of effective platform engineering, enabling developers to access the resources they need without having to rely on operations or infrastructure teams. This autonomy reduces bottlenecks because you don't need approval for things like creating new staging environments. Developers can simply start isolated environments with a simple command, thereby maintaining productivity without deep infrastructure knowledge.



How do platform engineering and DevOps differ

Platform engineering and DevOps have distinct objectives and different roles and responsibilities. They also affect developers in different ways.

Focus and purpose

Platform engineering focuses on building and maintaining internal platforms to streamline application development and deployment. In contrast, DevOps seeks to integrate development and operations to enhance collaboration, automate workflows, and accelerate delivery cycles.

- DevOps involves concepts, methodologies, and best practices proven to improve the software delivery experience.
- It focuses on automation, autonomy, and communication — advocating solutions that can support modern workflows.
- Platform engineering focuses on building and maintaining the IDP, allowing developers to engage immediately in their core work.
- With platform engineering, you approach the platform's evolution in the same as you would that of other products, treating developers as customers.

Roles and responsibilities

Platform engineers are responsible for creating and managing scalable, reusable platforms while overseeing infrastructure, tooling, and developer experience. DevOps engineers bridge the gap between development and operations, managing CI/CD pipelines, monitoring, and automation to ensure continuous and smooth delivery and deployment.

- DevOps specialists create high-level processes that enhance collaboration between development and operations teams.
- DevOps responsibilities cover the entire software development lifecycle, including early stages like requirements planning.
- Platform engineering involves creating a dedicated team that builds developer tools using a platform-led approach.
- Platform teams work closer to the infrastructure, creating effective self-service tools that empower developers.

Impact on developers

Platform engineering provides developers with stable, scalable, self-service platforms, reducing infrastructure management overhead. This allows developers to focus on writing code and developing features. DevOps fosters a culture of collaboration and shared responsibility, enabling developers to be more involved in deployment and operations, enhancing overall agility and responsiveness.

- CI/CD pipelines, test suites, and IaC are commonly found in a DevOps context but can be challenging for developers to utilize.
- Developers often lack the resources to develop DevOps processes themselves, as their primary role is to complete the development tasks that drive business value.
- Platform engineering aims to make life easier for developers by providing the necessary tools and workflows.
- The IDP lowers cognitive load, increases fulfillment, and allows developers more time for meaningful progress on new product features.

The shift to platform engineering

At its core, platform engineering recognizes that the ultimate goal is not simply delivering infrastructure for developer teams but supporting developers in a scaled-up organization.

This encompasses many original DevOps philosophies, adding several technologies and methodologies to create a holistic development experience:

Automation and IaC — Infrastructure should be automated and reproducible. IaC tools define what the platform should look like and enable new instances to be created on demand. Manual actions are minimized to eliminate friction points in the development flow.

Focus on efficiency — The platform should be designed to solve the most common challenges encountered by developers. Instead of trying to recreate functionality that already works well, focus on supporting the unique needs of your teams. Rolling out your own CI/CD or source control system is unlikely to be beneficial, for example, but providing a mechanism that mirrors a snapshot of your production infrastructure into a fresh staging environment could save developers hours each week.

Self-service access — Every part of the platform should be an asset that developers can freely utilize. You're providing a toolbox of controls for developers to use as they see fit. Avoid prescribing specific usage patterns, as individual engineers may work in slightly different ways.

Continual evolution — The platform should be continually developed using the same product-driven mindset you apply to customer-facing functionality. Although the "customer" is internal developers, it's still vital that improvements are implemented promptly to ensure the platform effectively meets their needs. This keeps developers productive over a sustained period of time.

Each of these principles centers on simplifying the development experience. Platform teams must listen to development teams and then provide the toolchains they require.

Whereas DevOps roles typically have a broad remit and can include many different responsibilities, platform engineers focus exclusively on the creation of IDPs. These platforms are designed to act as the development team's operational center, offering everything needed to deliver quality software on time:

- IDPs provide the mechanisms that enable self-service, on-demand access to infrastructure, such as by offering custom CLIs and web interfaces that let developers connect to the resources they need.
- Platforms include capabilities to help developers set up new environments and efficiently test their changes.
- When all developers use the IDP for their work, team leaders can continually and centrally enforce critical security and governance policies.

Hence, creating an IDP improves the developer experience and delivers consistency throughout the delivery lifecycle. This facilitates productivity increases and can give the business a competitive advantage by allowing it to launch new features quickly without affecting quality standards.

Benefits of platform engineering

Organizations can be hesitant to invest in platform engineering. A common concern is whether the engineers working on the platform would be better utilized within the main product team. Here are four reasons why you should commit to platform engineering.

Accelerating development

Internal platforms increase development speed by giving developers automated processes and self-service infrastructure to make them more productive. They can make continuous progress on the product features the business prioritizes.

Once a feature is ready, the development team can spin up a new test environment to verify the change autonomously. The platform can perform automated tests and then ship the feature to customers in production while developers start work on the next task. This reduces time to market without compromising on quality.

Promoting focus and specialization

Developers should be able to focus on their key proficiency — development. Modern infrastructure, CI/CD, and distributed deployment systems are dedicated specializations. Developers don't need to be experts in these fields and may sometimes struggle to understand them, preventing them from retaining their focus on building new software. The platform engineering team will include experts skilled in relevant topics such as IaC, CI/CD, and PaaS solutions, providing the specialization required to speed up advancements in both development and infrastructure.

"Spacelift provides the kind of tooling that supports governed decentralization, enabling and empowering developers to solve problems themselves — even for a very regimented company. You could build Spacelift policies that are very restrictive of what a developer can do, so even in a PCI-regulated environment much like Checkout.com, you can still have teams that are empowered in their IaC to use these kinds of tools and innovate."

Joe Hutchinson

Director of Engineering — Developer Platform
Checkout.com

Ensuring tools and processes continually develop

Development processes need to evolve as your product grows. Over time, your stack will add new technologies and requirements. You might introduce a new storage system, require more comprehensive end-to-end tests, or have to comply with another regulatory standard.

Platform engineering ensures your toolchain develops in tandem. Without it, you have to make ad-hoc workflow adjustments, which can be poorly documented and difficult to maintain. Furthermore, developers often don't have time to make the optimizations they require, so inefficient practices continue even after they've been recognized as bottlenecks. Giving devs access to a platform engineering team allows frustrations to be addressed without delaying the product's release schedule.

Improving developer experience (DevEx)

Platform engineering improves DevEx through regulated self-service infrastructure. Developers who require infrastructure provisioning to test their applications can easily use the self-service mechanism the platform team provides, without requiring any other input from them to test their application. This reduces the time spent on testing new features, resulting in faster deployment and time to market.

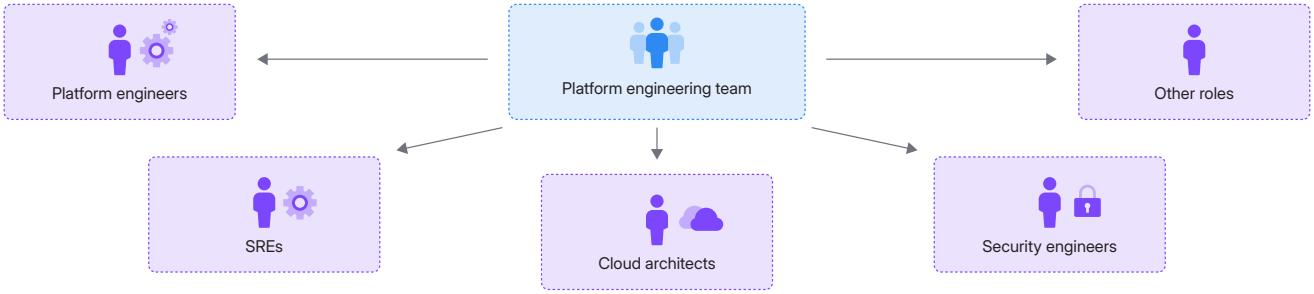
"Confidently delegating much of IaC management to the individuals that own it allowed our teams to transform into service teams — not just remote hands gatekeeping AWS."

Maxx Daymon

Staff Cloud Platform Engineer
1Password

Building your platform

Your platform engineers should work closely with SREs, cloud architects, and security engineers to build a successful platform. Platform engineers should be experts in IaC (OpenTofu, Terraform, Pulumi, etc.) and cloud technologies (AWS/Azure/Google Cloud, depending on the cloud provider the organization is using). They should also monitor the platform's reliability, scalability, and performance while ensuring the necessary standardization and governance. This means they should have expertise in CI/CD, monitoring, observability, governance, and compliance tools.



Team roles and responsibilities

Platform engineers will discuss challenges with development teams and act upon their insights to build internal platforms. Their tasks can include the following:

- Configuring IaC tools to provision new infrastructure on demand
- Working with existing infrastructure and operations teams to bridge the gap between dev and prod
- Implementing and maintaining CI/CD pipelines that automate inefficient workflows
- Creating bespoke internal tools to accommodate org-specific workflows, enforce security policies, and maintain compliance with regulatory standards
- Building, maintaining, and documenting custom APIs, CLIs, and web UIs that expose the platform's functionality — for example, an API that exposes the number of errors in each environment or a CLI that pushes local code straight to a new sandbox.

Building your toolstack

Platform engineering encompasses cloud services, containers, automation, monitoring, and other areas, so it covers many tools:

Type of tool	Examples
Cloud Services	AWS, Microsoft Azure, Google Cloud, Oracle Cloud, etc.
Version Control Systems	GitHub, GitLab, BitBucket
Infrastructure as Code (IaC)	Terraform, OpenTofu, Pulumi, AWS CloudFormation
IaC Orchestration Platforms	Spacelift
Containerization	Docker, Podman
Orchestration	Kubernetes, Docker Swarm
Configuration Management	Ansible, Chef, Puppet
CI/CD	Jenkins, GitHub Actions, CircleCI, GitLab CI
Monitoring	Prometheus, Grafana, ELK Stack (Elastic Search, Logstash, Kibana)
Security	Open Policy Agent, AWS Secrets Manager, Vault
Programming Languages	Python, Golang, Bash, Powershell

Platform engineering best practices

Bear these practices in mind as you adopt platform engineering:

1. Store your code in a VCS – This improves collaboration, tracks changes, and can facilitate reverting to a previous state when errors appear.
2. Adopt IaC – Automate and provision your infrastructure to reduce manual repetitive costs and minimize human errors.
3. Implement CI/CD – Reduce deployment times, automate builds/tests, and ensure reliability.
4. Increase observability – Monitor the health and performance of your applications and infrastructure. Ensure logs are collected and easily accessible for troubleshooting.
5. Improve security – Use the least privilege principle, manage secrets securely, and scan regularly for security vulnerabilities.
6. Build for scale – Design your infrastructure for scaling out rather than scaling up (this will ensure you add more instances to your workload, rather than more resources for one instance) and also design for failure by implementing high-availability and disaster recovery mechanisms.
7. Optimize usage – Ensure your resources are used efficiently and optimize costs.
8. Improve documentation and knowledge-sharing – Promote a culture of collaboration between teams and share the necessary resources to understand the architecture, configurations, and processes.

How platform engineering enables self-service infrastructure

As organizations scale, collaboration between teams becomes more complex. Enabling autonomy for development teams and reducing bottlenecks in platform teams is vital. A key plank of platform engineering is making every part of the platform freely available to developers, so providing self-service access to infrastructure is central to its success.

Establishing a self-service culture is not easy, but the benefits repay the effort in a very short time. As the organization scales, a strong self-service culture makes it easier to cooperate and improve the quality of delivered solutions to both internal and external clients.

Here are the main benefits:

Organizational lens

- Increase autonomy. The team (i.e., the development team) decides the infrastructure to be created. Their decisions are based on available templates, guidelines and collaboration with the self-service infrastructure team (i.e., the platform team).
- Decrease waiting times. The development team doesn't have to wait for infrastructure resources.
- Increase productivity. With increased autonomy and decreased waiting times, the development team can reduce waste during work.
- Leverage knowledge-sharing. By enabling teams with self-service, the organization can share best practices and standards more widely.
- Respond to the shortage of infrastructure, network, system, SRE, or DevOps engineers. IT talent is in high demand. Having a structured, organized team of experts who act as a platform team helps the organization to avoid over-hiring and allows them to better manage people and resources.

Processes and engineering lens

- Control and decrease the costs of infrastructure
- Unify stacks and used technologies
- Implement and control security throughout the whole organization
- Control integrations, i.e. authentication services

How does Spacelift enable self-service infrastructure?

Spacelift is an infrastructure orchestration platform that meets the requirements for self-service infrastructure we've discussed above. It delivers all the features the platform team may need to provide their services to internal and external development teams.

1. Maximizes reusability

A key benefit of self-service infrastructure is the potential to use predefined, widely available templates for different use cases. Spacelift implements this functionality through Blueprints.

The platform team defines the blueprints and uses Spaces and Policies to allow other teams to use them. Blueprints can be as simple as creating an EC2 instance in AWS or as complicated as setting up a network using a virtual private cloud (VPC) and connecting it to other existing networks through a transit gateway or virtual private network (VPN).

With parametrization of Blueprints through Inputs, platform teams have a very powerful toolset to deliver business-focused, flexible solutions that are fully in their control.

2. Prioritizes security

One of the benefits of self-service is enhanced security. Allowing multiple teams to use a vulnerable infrastructure template creates unacceptable risk. However, expecting every team to create and deploy their own infrastructure leaves you open to deploying vulnerable infrastructure.

You can mitigate both risks by implementing the Spacelift platform using a self-service approach. The platform team designs and prepares Blueprints, which should be subject to extensive testing before publishing.

Spaces help to ensure that the blueprints will be deployed on permitted environments only. For example, if the organization uses a CI/CD account, all templates related to CI/CD should be deployed there, not in the production account. Spaces can be used to represent the cloud setup, for example accounts, subscriptions, or environments.

Another layer for securing Blueprints is policies. Policies can be attached and enforced in published blueprints, so the platform team can ensure specific behavior and specific use of the templates.

"To provide a catalog of infrastructure components, we took advantage of OPA integration to automate policy and implement policies directly in code which allowed us to set guardrails on what users can and cannot do thus ensuring infrastructure consistency and best practices."

Timur Bublik

Platform Engineering Lead
Tier Mobility

3. Enhances efficiency

Blueprints are also a key concept to remove the blockers that can make teams less efficient. The development team doesn't need to focus on acquiring additional skills within the team, such as expertise on Terraform, cloud infrastructure deployments, etc. Instead, they can use Blueprints to deploy specific infrastructure to specific environments.

4. Enables CLI and API

Using CLI tools (or API calls) allows you to build wider solutions, in which a specific component is triggered by an overarching service. This can happen when the platform team more complex solutions, as well as self-service infrastructure.

Spacelift has its own Terraform/OpenTofu provider that can be leveraged easily to manage any Spacelift resource. It also offers a CLI called spacectl, a flexible tool that is easy to install, as well as the GitHub Action to install and configure it inside the CI/CD pipeline. And that's not all: Many engineers prefer to use APIs in certain cases, so Spacelift offers the GraphQL API, which allows users to interact with a service in a programmatic way.

5. Boosts usability

Creating the best possible approach to self-service infrastructure often raises a dilemma. Cloud setup in modern organizations is complicated and crafted exactly for specific needs. Use of templates reduces this adaptability, so the platform team has two options: Create complex templates and push teams to use the single solution, or create many small templates and ensure that teams are able to construct complicated solutions with them. The latter requires the team to have at least an understanding of IaC and self-service technologies.

Spacelift solves this dilemma with **stack dependencies**. This approach allows platform teams to deliver well-designed, single-purpose solutions. Their clients select the blueprints that best fit their needs and connect them into logical chains, using stack dependencies.

6. Enforces best practices

One of the platform team's duties is to implement and enforce best practices. This remains true when you apply a self-service infrastructure approach. By combining Spaces, Blueprints, policies, and the module registry with a modern self-service approach to manage the dependencies between stacks, it becomes easier to ensure best practices are followed.

A compelling feature of Spacelift is the way it allows you to manage the entire platform setup. As mentioned before, Spacelift has published its own **Terraform provider**, which means the process of managing the platform governs all processes in the organization, such as infrastructure coding (and creation), version control, CI/CD, security scans, governance, etc. This approach allows the organization to achieve next-level quality and cooperation.

In **this article**, we explore examples of self-service infrastructure, the problems it solved, and the improvements Spacelift can add.

“The primary win has been related to eliminating the manual process of direct Terraform applications and coordinating changes between engineers. We have been able to lower the use of direct privileges for deployment without changing our ability to develop IaC.”

Chris Schafer
Senior DevOps Engineer
Archipelago

Real-world self-service infrastructure implementations

We will now walk you through some real cases we've observed or worked on:

Case 1: CloudFormation for setting up environments

This project centers on the team who managed the infrastructure templates for development teams. They used AWS CloudFormation to create separated environments in multiple AWS accounts. The infrastructure created in these environments contains networks, policies, container clusters, serverless, and more.

The problem

The first problem in this case concerned knowledge gaps around AWS architecture and best practices within development teams. The second was more complex: The organization had numerous development teams, with each team owning at least three AWS accounts, sometimes more. This proliferation of teams and ownership meant there was little control over the infrastructure and no single pane of glass where information about systems could be gathered.

How it was addressed

The initial solution was to create a set of CloudFormation snippets that could be collected by the development team and combined into one big template according to their needs.

What it solved	Issues raised by this approach
Centralized repository of the CloudFormation templates prepared by experienced engineers	The decision to use small snippets led to a huge number of files. Development teams always needed to have help with selecting proper templates.
Way to keep infrastructure under specification designed by DevOps team	Removing the responsibility of creating the full template from snippets didn't eliminate the infrastructure workload for the development team. They still had to know how to work with it.
	Although snippets were versioned and stored in the repository, development teams were still in charge of creating the final template. This meant that the development team could still introduce misconfigurations into infrastructure.
	Disconnected steps in the process meant the platform team didn't know who is using snippets and the development team needed to constantly monitor the updates.

It is clear that this approach did not deliver a satisfactory solution, failing to resolve technical issues or improve the process.

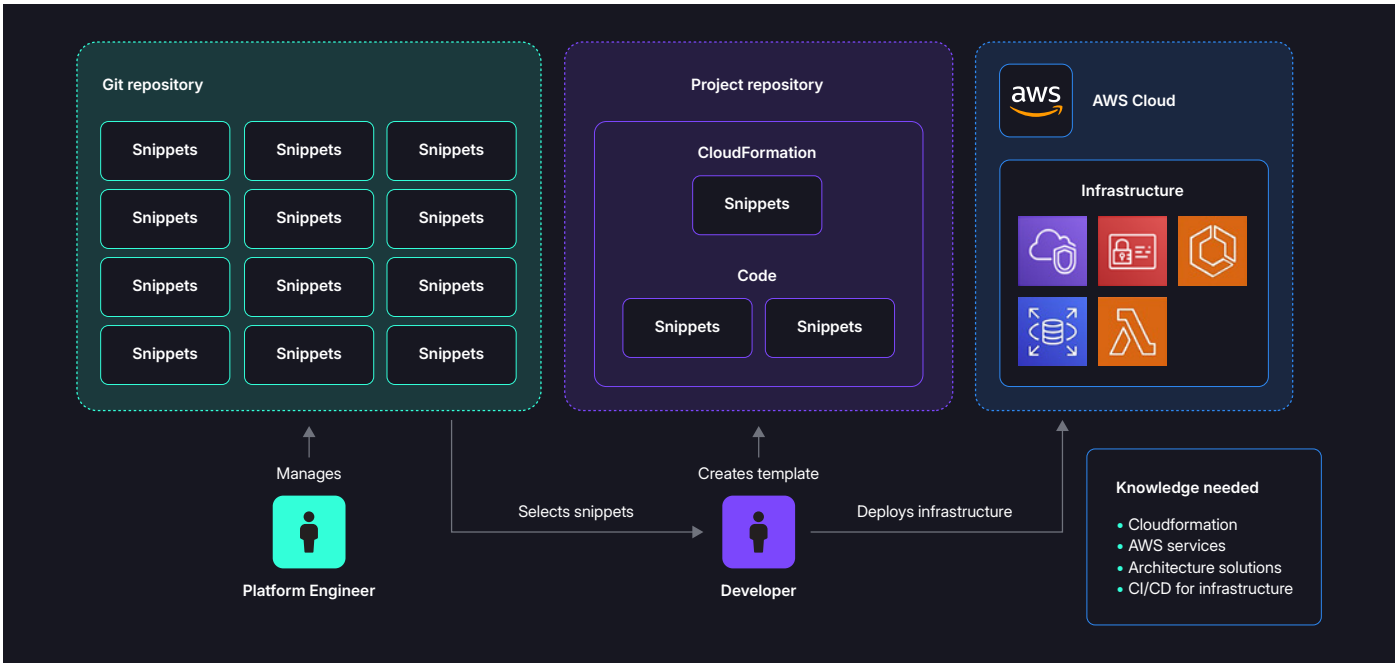
How Spacelift could solve the problem

The first step would be to devise an appropriate process for defining the requirements and designing the templates. With Spacelift, you don't need to use unwieldy CloudFormation snippets: you can prepare and parameterize full templates. For optimal flexibility, the templates should be prepared per their layers. For example, you would have separate templates for the VPC, ECS cluster, and so on. Finally, the templates can be published as Blueprints.

We mentioned the complicated AWS account setup earlier. This setup can be replicated using Spaces and guarded by policies.

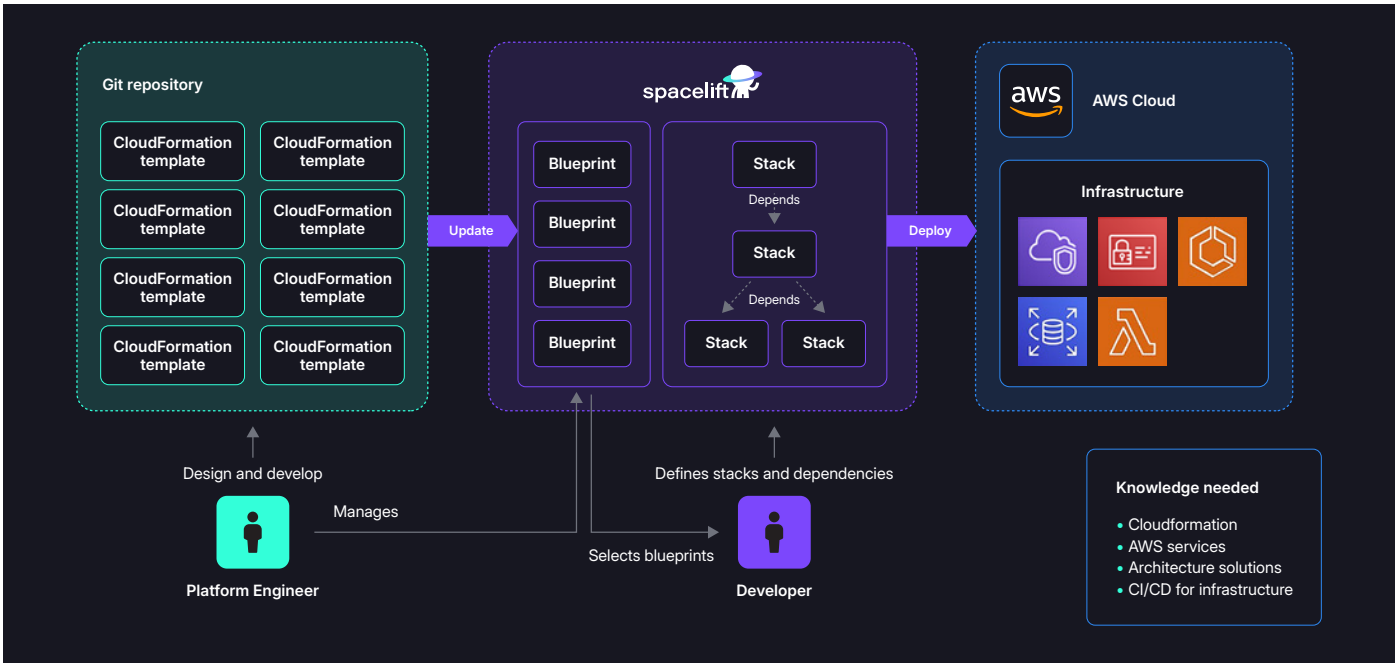
When the development team creates their stacks from Blueprints, they use stack dependencies to connect all stacks in a logical chain of deployment.

The first diagram below presents the company's process. The second shows the process with Spacelift. We also show additional skills the team needs to work with these processes.



This diagram shows the process as is. The development team has to do considerable manual work on infrastructure tasks to create the templates properly. The platform team doesn't know who is using the snippets (and how).

Even worse, the development team has full access to the snippets, and they can modify them. These issues add tension and uncertainty to the process.



The second diagram shows the Spacelift approach for self-service infrastructure. It clearly separates responsibilities and uses one central tool. The respective responsibilities of the platform and development teams are clearly defined. There is an established control and communication channel.

Development teams can focus on their main goal — delivering software. The platform team controls the CloudFormation blueprints and these cannot be modified by development teams, enforcing configuration best practices.

The platform team can also easily track the quality of executions and improve the blueprints when needed.

Case 2: Multicloud Kubernetes

Kubernetes shines as an agnostic tool that can be used in a range of applications. It can be deployed on AWS and Azure or AWS and on-prem, for example. Such multicloud setups are useful, especially when different workloads operate under different security guardrails.

In this example, we explore creating self-service infrastructure for such a scenario.

The problem

This is a fairly straightforward case, in which the development team wants to quickly deploy a workload on a Kubernetes cluster. Depending on security recommendations, deployment will take place in AWS or on-prem.

How it can be solved

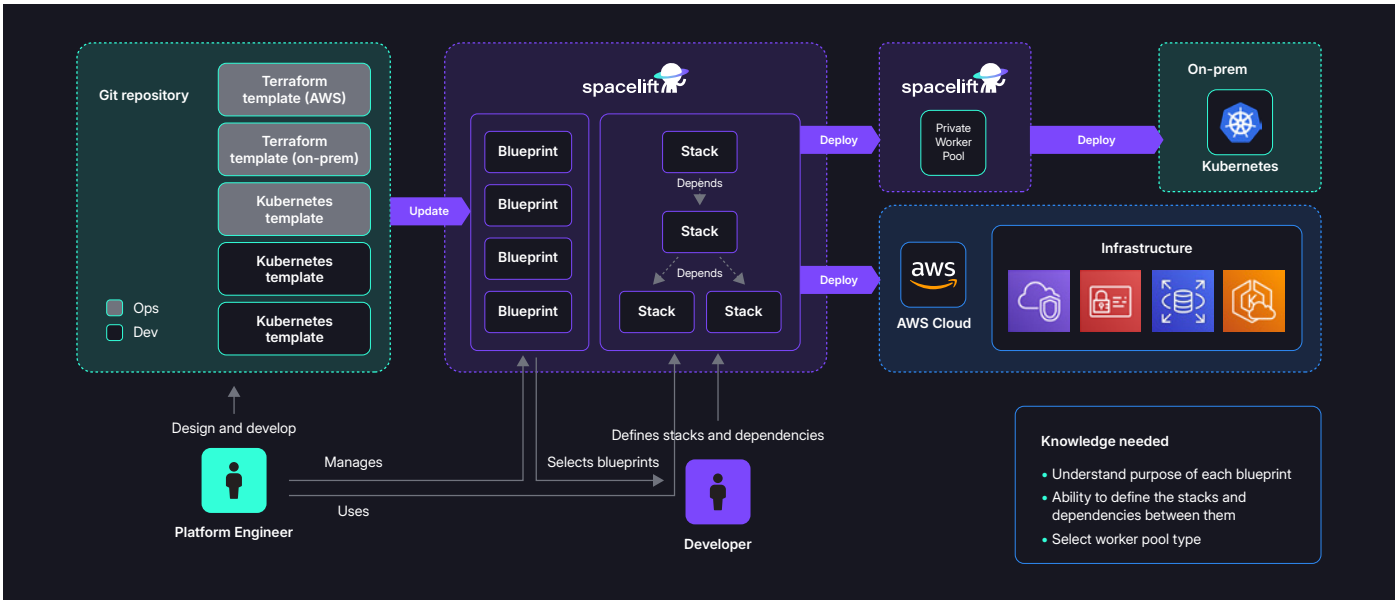
For this exercise, we assume the platform team is in charge of Kubernetes infrastructure and keep the solution simple, with one AWS account per environment and one Kubernetes cluster. The same setup is on-prem — one cluster per environment.

Let's explore the responsibilities of platform and development teams in this scenario:

Platform team responsibilities	Development team responsibilities
Build blueprints for cluster's infrastructure	Create stacks from blueprints and configure workload
Create and manage clusters	Select proper Kubernetes cluster for deployment
Create and control proper separation between clusters	Although snippets were versioned and stored in the repository, development teams were still in charge of creating the final template. This meant that the development team could still introduce misconfigurations into infrastructure.
Build blueprints for development teams to deploy workloads	Disconnected steps in the process meant the platform team didn't know who is using snippets and the development team needed to constantly monitor the updates.

As we can see, the platform team is responsible for the underlying infrastructure. Fortunately, this responsibility can be executed through Spacelift, in the same way that development teams use it.

To avoid confusion for development teams, Spacelift administrators create Spaces that logically separate platform team administrative work from development work. Another set of Spaces can be created to separate environments and clusters (on-prem, AWS). This logical structure must be part of SDLC design.



The diagram above illustrates the potential process of this solution. It offers simple and compelling benefits:

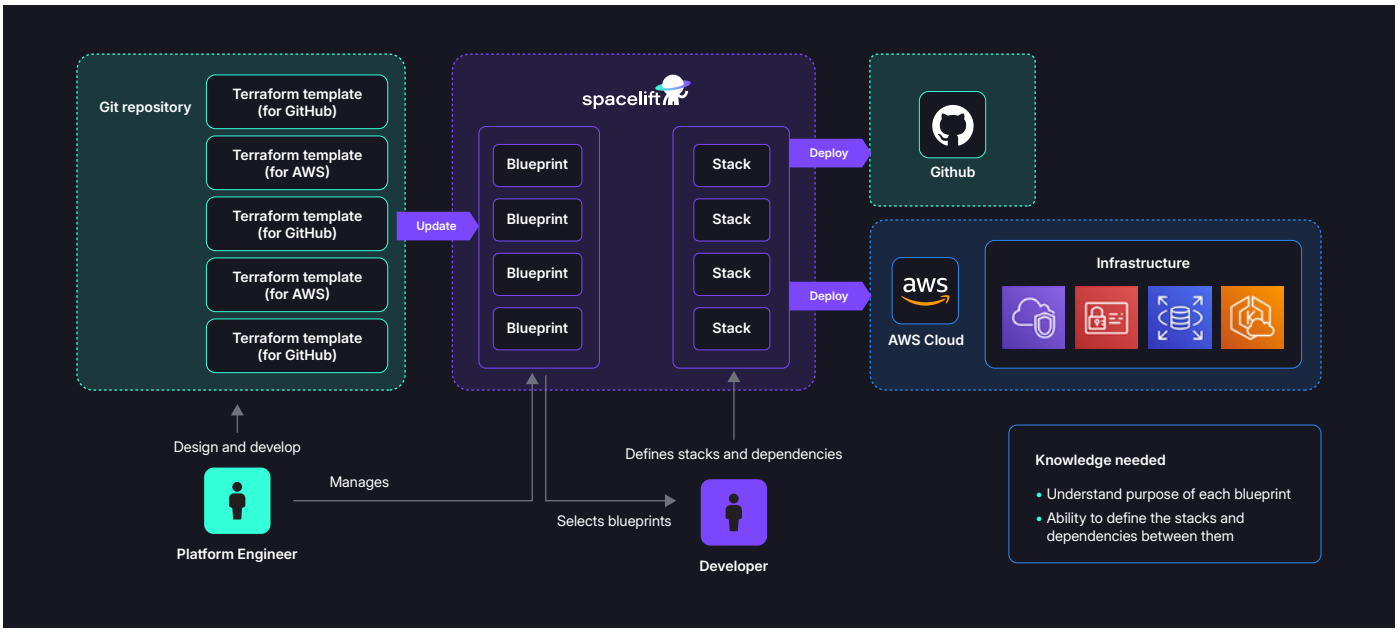
- All decisions and deployments are made with one tool (Spacelift).
- A single team is responsible for infrastructure.
- Developers need to know only where to deploy workload. From their perspective, the additional work is to select a defined worker pool.
- No additional knowledge is expected from development teams.
- Development teams are not blocked with their deployments.
- Spaces separate the responsibilities of platform and development teams.

Case 3: Create entire project for development teams

In this case, we look at a real-world scenario in which the platform team sets out to deliver fully functional templates for development teams to create projects. These templates must contain infrastructure on AWS, as well as GitHub repositories and pipelines. We will approach this case in two steps. We have already created an appropriate setup in Spacelift, with Spaces, etc.

The first step is to create and configure the GitHub repository. Configuration involves assigning proper users, configuring the protected branches, and so on. The second step is to deploy infrastructure on AWS. This step is strictly related to infrastructure.

This is how the process might be constructed:



The platform team creates the templates for all use cases. There may be just one template related to GitHub repository creation, enabling almost complete control over the way repositories are constructed and managed. This is a very important consideration when your organization is scaling. The approach allows you to keep the SDLC approach unified for the whole organization, which can be important from a regulatory perspective.

This is a very straightforward step for the development team,. They simply select the name for the repositories and who should have access to them. They are then ready to ship their code into the repository.

The second step is similar to previous scenarios. The platform team prepares and manages blueprints for infrastructure, and development teams use these blueprints to create their infrastructure for applications.

"It's much easier for the developers to synchronize releases between the infrastructure code and the developer code. Before, we were managing the steps to do that, but right now Spacelift does that for us, by default or via policy. This new way of managing Terraform helps us improve collaboration and makes the developer autonomous."

Kévin Lemele
Senior Platform Engineer
Payfit

Conclusion

DevOps and platform engineering have transformed the process of delivering and deploying software by allowing developers to focus on their key specialty. As forward-looking organizations transition from a centralized DevOps approach to platform engineering, a successful self-service system becomes pivotal for enhancing developer velocity with control.

You won't establish a well-functioning self-service culture overnight, but the results will transform your developers' productivity. As your organization scales, having a strong self-service culture in place will make it easier for teams to cooperate and ensure that the solutions are of the highest quality and are delivered to internal and external clients quickly.

A specialized infrastructure orchestration platform like Spacelift can be your secret weapon as you consolidate a self-service culture in your organization. Maximize reusability, prioritize security, enhance efficiency, enable API and CLI, boost usability, and enforce practices with features like Blueprints, spaces, policies, stack dependencies, and Spacelift's own Terraform/OpenTofu provider.

By enabling on-demand infrastructure orchestration capabilities, Spacelift will help your organization remain agile and competitive in a technology landscape that never stays still.

About Spacelift

Spacelift is an infrastructure-as-code (IaC) management platform for orchestrating the full lifecycle of your infrastructure. It integrates with your choice of VCS to access and manage your infrastructure code across all IaC tools (e.g. Terraform, OpenTofu, CloudFormation, Pulumi). Spacelift workflows orchestrate the full lifecycle of your infrastructure — provisioning, configuration management, observability-tool and security-tool integration, cloud resource management, and container orchestration.

By providing developer self-service, golden paths with guardrails, and an OPA policy engine, Spacelift empowers businesses to accelerate developer velocity while maintaining control and governance over their infrastructure. Spacelift offers unrivaled support, no-nonsense pricing, and a range of deployment models to fit your specific needs.

Learn more about the Spacelift platform and how it can help you overcome your IaC challenges at **Spacelift.io**. Sign up for a **demo**, or test the platform yourself with a **free trial**.