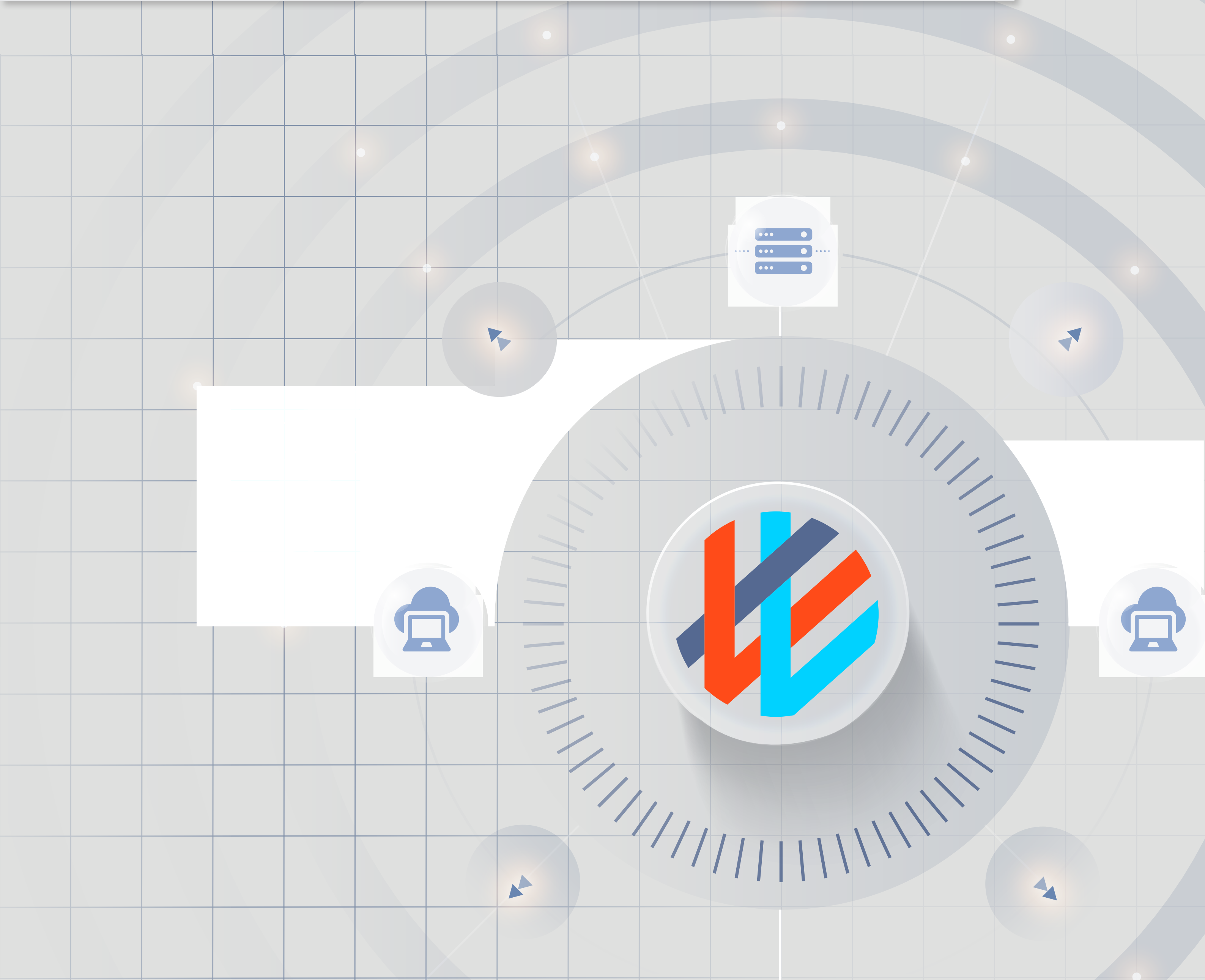
weaveworks

WHY SELF-SERVICE IS KEY TO DEVELOPER PRODUCTIVITY



Introduction

DevOps has become widely popular as it has met organizations' goals for application quality, team productivity, and cost optimization. However, there is a flip side to the story of DevOps' success. While DevOps aspired to meet the mantra 'you build it, you run it', this wasn't the case. In reality, developers ended up relying on Ops to manually provision and maintain infrastructure. To add to this complexity, they also have to manage multiple tools, frameworks, and repositories across the software lifecycle. This slowed down release cycles and exposed the other side of DevOps.

With the rise of cloud-native application development, application architecture has become more complex, as has the software supply chain. Kubernetes, though powerful, adds layers of new components and new ways of building and shipping applications. These challenges call for improving Developer Experience (DX) so organizations can accelerate delivery.

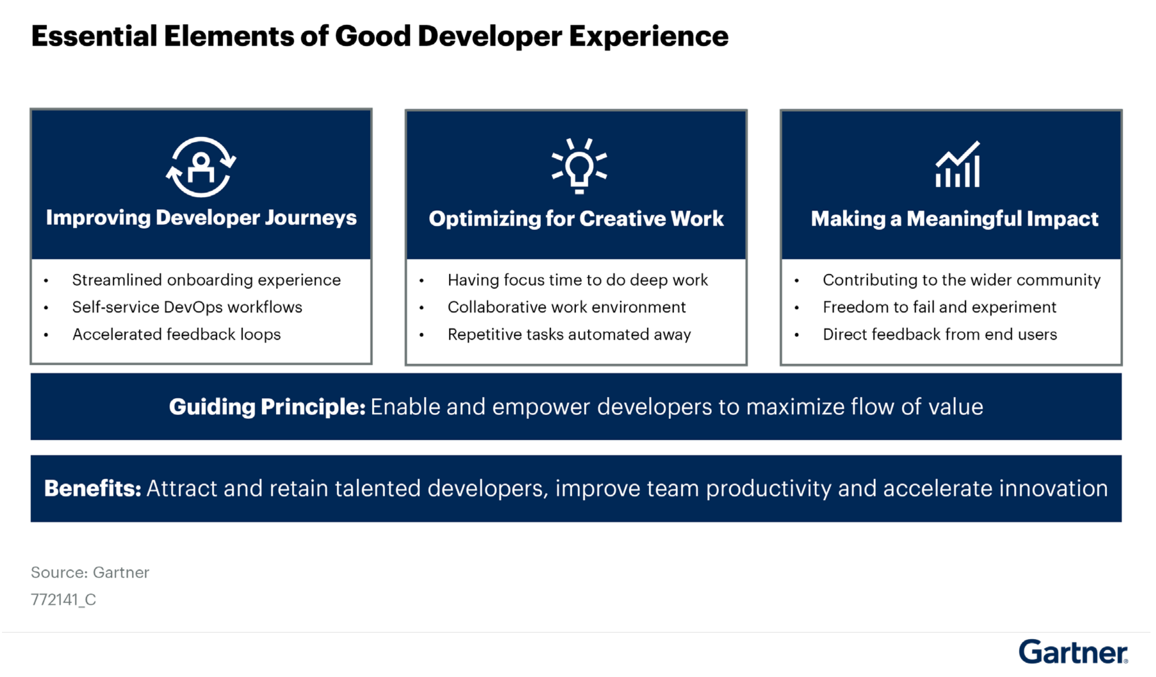
According to Gartner, "Developer experience refers to all aspects of interactions between developers and the tools, platforms, processes and people they work with, to develop and deliver software products and services."

This encompasses all activity from the time code is written to the time it is shipped to production. Developers are most productive when they have the autonomy to move quickly and seamlessly from concept to code to production. This can be achieved through a 'self-service developer platform' which, as the phrase implies, is all about empowering developers to easily and reliably release code with as little friction as possible. It's about automating and abstracting away routine Ops tasks. This makes developers less dependent on the Ops team to provision resources or move code through the pipeline. The result is that developers do not need to switch contexts and can focus on what they do best - writing code.

Gartner's Take on Self-Service Developer Experience

Developer self-service has the inherent benefit of bringing consistency and repeatability to otherwise disparate processes and error-prone manual handoffs. The goal of self-service is to ensure developers have an experience that makes 'the right thing to do, the intuitive thing to do.' For example, the ability to self-serve pre-vetted open-source libraries from a trusted component catalog improves governance, as well as developer experience.

As shown in the diagram below, a good developer experience is all about improving developer journeys, optimizing for creative work, and making meaningful impact. The end goal is to empower developers to ultimately improve team productivity and accelerate innovation.



Essential Elements of Good Developer Experience

Improving Developer Journeys
- Streamlined onboarding experience
- Self-service DevOps workflows
- Accelerated feedback loops

Optimizing for Creative Work
- Having focus time to do deep work
- Collaborative work environment
- Repetitive tasks automated away

Making a Meaningful Impact
- Contributing to the wider community
- Freedom to fail and experiment
- Direct feedback from end users

Guiding Principle: Enable and empower developers to maximize flow of value

Benefits: Attract and retain talented developers, improve team productivity and accelerate innovation

Source: Gartner
772141_C

Gartner

The Gartner report explains how poor developer experience can lead to issues like governance costs, non-uniformity between teams, and distraction from their actual focus on coding.

Gartner Report, Software Engineering Leader’s Guide to Improving Developer Experience, 2022.

There is a growing awareness about the importance of developer experience, and how to design it so it doesn’t hinder but enables developers to get more done. Let’s start with examining what developers don’t deem as ideal.

What a Bad Developer Experience Looks Like

While DevOps was conceptualized to eliminate the siloed development process by unifying development and operations teams, it did little to scale development. Organizations struggle to ship products faster due to the constant back-and-forth between the teams and a further blockade of approvals.

When it’s time to deploy the application a developer has built, they need the appropriate resources and environments provisioned. It involves multiple stakeholders and can take days to finally have the resources provisioned. Friction between Developer and Ops teams often arises. Developers are tasked with producing new and innovative products quickly and Ops teams are tasked with maintaining reliable infrastructure while meeting the demands of all stakeholders.

1 **Developers are frustrated with manual processes**

Traditionally, developers follow ticket-based ops, which can take multiple days before the team provides the resources. It involves the development team raising a ticket for resources, which are assessed by Ops and forwarded to stakeholders for approval. In some cases, meetings involving several stakeholders are held to explain and justify the need for the required resources. The Ops team needs to conduct their own due diligence to consider whether there is proper monitoring and security guardrails for these new resources before they are provisioned. More mature organizations set Service Level Objectives (SLOs) via SRE best practices to ensure no surprises later. All of this back and forth can take weeks, not to mention the friction it can cause between developers and the Ops team.

Following this elaborate process of gaining all approvals, the resources are finally provisioned. The entire process is draining on developers and hinders them from their main objective - writing code and delivering value to customers. Ops teams are stretched thin catering to the various needs while doing their best to keep the systems compliant and reliable.

2 **Developers drown in tool sprawl**

With complex processes come numerous tools as well. Developers are forced to manage issues that arise from the use of highly complex tools that are held together by duct tape integrations. They are tasked with controlling and managing cloud-native infrastructure while their core competency is writing code and building software. Consequently, things break all the time, and developers end up in firefighting mode rather than getting work done.

3 **Kubernetes newbies get left behind**

While Kubernetes is seeing broad adoption across industries, one of the biggest challenges facing the cloud-native ecosystem is the lack of Kubernetes knowledge among developers. It can be quite a challenge to set up and maintain a Kubernetes environment and developers are not as familiar with this new technology as their Ops counterparts. As a result, many developers are forced to grapple with new jargon and concepts to speak the same language as Ops does. Deploying simple cloud resources could involve numerous steps and a steep learning curve. This again slows down developers and puts additional burdens on them.

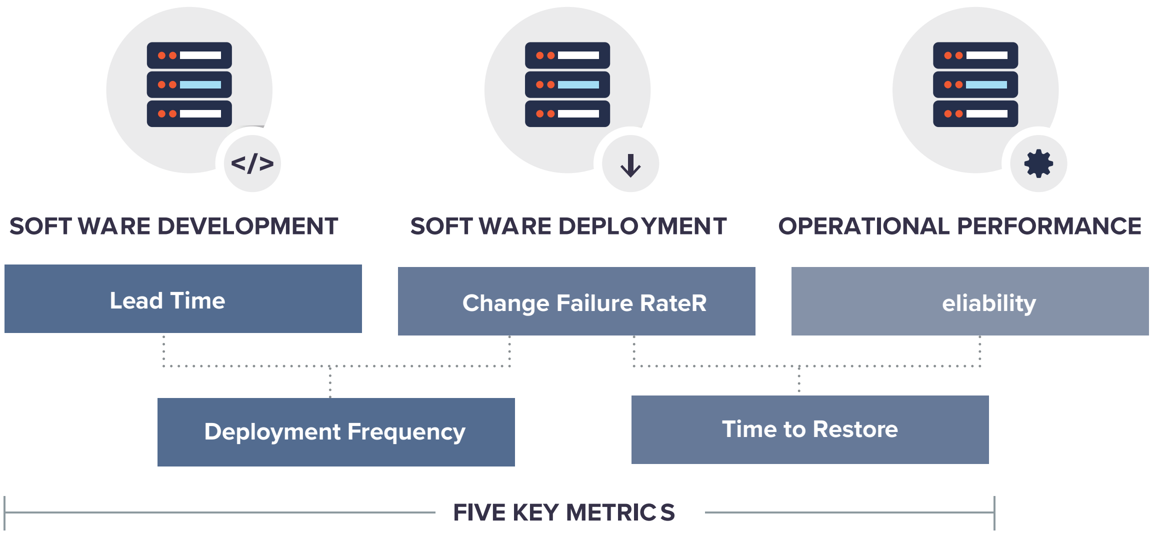
4 **Caught in a loop of work duplication**

Shoddy resource provisioning has real implications when an application reaches production. When apps and services fail in production, SREs and Ops teams come back to the development team to fix the issues that were caused due to incompatibilities, bugs, and resource constraints. Developers end up having to refactor or rearchitect parts or even the entire application and end up doing the same work all over again. This duplication of work is frustrating for developers and eats into their productive hours. The end result is application delivery is greatly delayed and lower-quality code makes it to production.

What Makes for a Great Developer Experience?

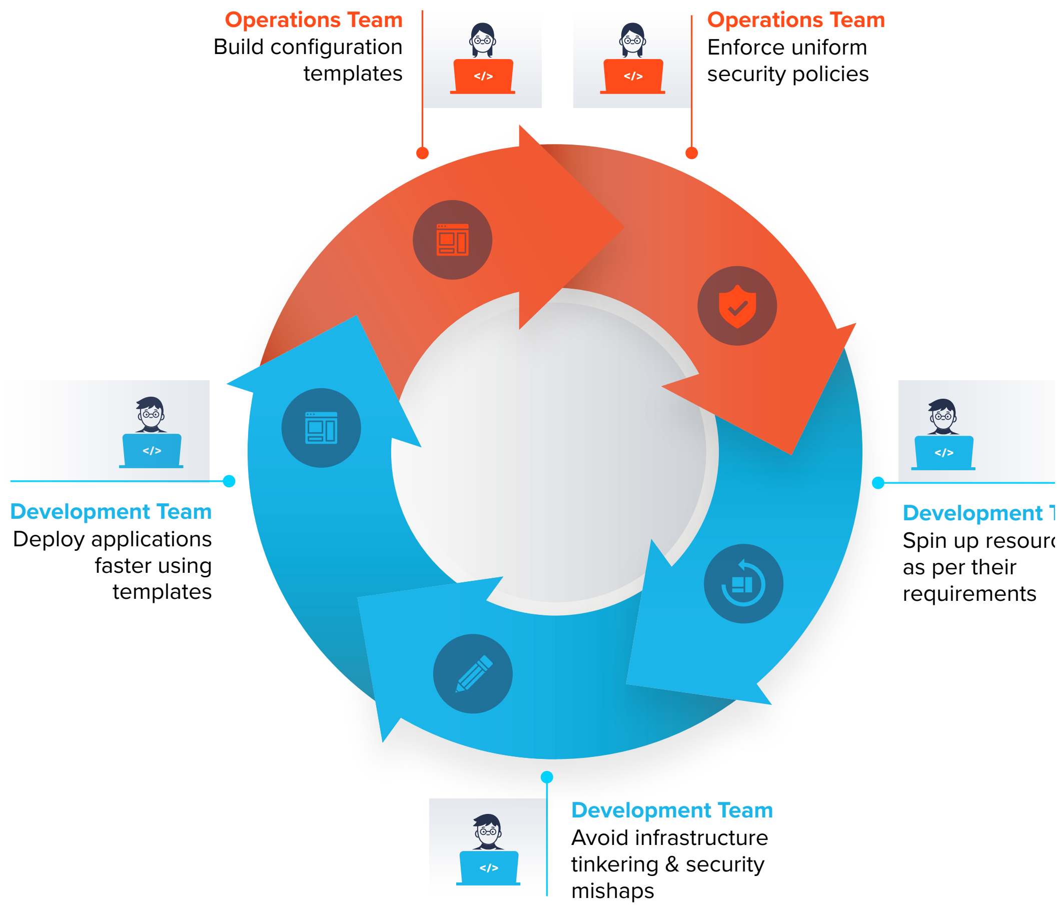
The **2022 DORA report** has identified five metrics that indicate the speed and effectiveness of your software delivery. The five measures are Deployment Frequency, Lead Time for Change, Mean Time to Recovery, Reliability, and Change Failure Rate. Of these metrics, the first two can be a measure and indicate a good developer experience.

Developers want to be able to deploy changes frequently, and greatly reduce the friction and time between the commit and deploy stages of the supply chain. In order to do this, it takes building a highly usable platform that reduces context switching, minimizes decision fatigue, and inherently includes speed of development, safety, innovation, and iteration.



According to the report, 63% of organizations have adopted internal platforms as a way to enable greater developer autonomy. An internal developer platform empowers Ops teams to make resource templates readily available for development teams. Such a platform would give developers the ability to focus on writing code, and deploying this code more frequently - essentially, a better developer experience.

The Self-service Model


Operations Team
Build configuration templates
Operations Team
Enforce uniform security policies
Development Team
Deploy applications faster using templates
Development T
Spin up resourc
as per their requirements
Development Team
Avoid infrastructure tinkering & security mishaps

Unlike the traditional ticket-based model, with a self-service developer platform, platform engineers can access the resources developers need available through pre-created and pre-configured services. These services are hosted on an internal developer platform, and this platform is continuously maintained and managed by the Ops team. In essence, the platform is treated like a product in its own right.

Implementing a Self-Service Development Platform

In a Kubernetes-centric world, building a self-service developer experience will need to take into account the ecosystem and practices that are prevalent in this particular ecosystem. Additionally, as DevOps evolves, it has given rise to newer practices like GitOps that are better at meeting the needs of developers. Let's look at how GitOps enables you to build a self-service developer experience.

GitOps: A Self-Service Model

According to the GitOps approach, the infrastructure layer, which is powered by Kubernetes, is defined in Git in the form of YAML files. YAML files define the number of clusters, pods in a cluster, resource limitations for each pod, and cloud-vendor-specific configuration as well. Using GitOps, the ops or platform team can create a resource template that is defined in YAML and that allows developers to deploy an application on, say, AWS EKS with a certain amount of memory and compute capacity. This template can include a commonly used toolset made up of a service mesh, a monitoring agent, a database, or any other component a typical deployment would require. These resources are pre-vetted by the platform team and are sure to work well with each other.

Meanwhile, developers can quickly access the resources they require in minutes without having to learn YAML. The use of the resource template abstracts complexity for developers and the intervention of the platform team is required only on rare occasions where the customization needs of your developers are not met. The reduction in time for provisioning resources from weeks to minutes is significant. It is only possible by eliminating manual decision-making and effort in the process and replacing it with software agents and policies that are transparent and predictable.

An important part of the GitOps approach is declaring the platform infrastructure and automating it using Git repositories. The platform is built, versioned, deployed, and managed using the configuration in Git. Any changes are made to the repositories and are automatically 'pulled' into production. Any configuration drift that occurs is automatically highlighted and corrected to the original desired state as described in the repository.

There are many benefits of a self-service developer platform that is powered by GitOps. The table below highlights the key differences between the traditional approach and a GitOps-based self-service developer experience:

GitOps-based Self-Service Experience	Non-Self-Service Experience
Resources conjured in minutes	Takes days to provision resources
No dependency on Ops teams	Ops team burdened with tickets
Consistent process across development team	Different process across development teams
Gives developers autonomy	Makes developers frustrated & helpless
Applies automation to ease process	Complicates process with manual effort
Scalable and in-built security policies	Unscalable manual process security measures
Inherent compliance audit	Compliance is a separate process
MTTR is minutes	MTTR is days
Accelerates deployment speed	Slows down software delivery

GitOps allows you to create a self-service experience for your developers that encourages them to innovate faster, reduce toil, and safely deliver new code through the security checks embedded by the platform team. It also reduces developers' dependency on the platform teams, thus minimizing friction between the two teams. Kubernetes runtime configurations against the policies and report back any violations.

Whitepaper: GitOps Accelerates Self-Service for Developers and Operators

Download this whitepaper and learn more about GitOps and Self-service platforms.

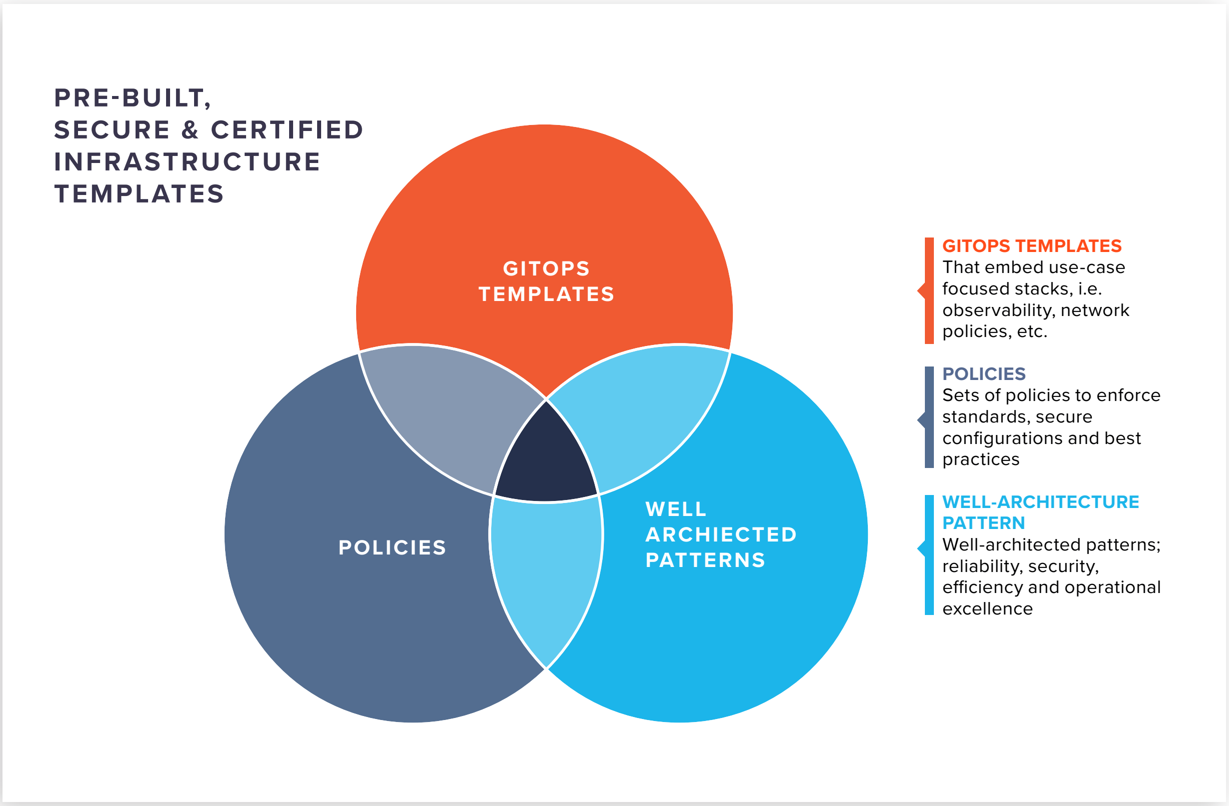
DOWNLOAD NOW

Build and Manage a Self-Service Developer Platform with Weave GitOps Enterprise

Weave GitOps Enterprise automates cloud-native app management to reduce human-driven errors and costs, while increasing developer productivity and operational efficiency. It is a continuous operations product that makes it easy to deploy and manage Kubernetes clusters and applications at scale in any environment. The single management console automates trusted application delivery and secure infrastructure operations on-premise, in the cloud, and at the edge.

Reduce Time and Errors With GitOps Templates

Weave GitOps eliminates the complex process of building an internal platform from scratch as it offers the foundational building blocks as features. GitOps templates allow you to template resources in a single definition; the resources can be anything that can be expressed in yaml (K8s, Flux primitives, TF controller, Crossplane, Cluster API). In other words, it allows you to define the various infrastructure and application components in the form of YAML configuration that can be deployed repeatedly and consistently. With WGE, platform teams can provision self-service templates that can be leveraged by application development teams. The template ensures there are required defaults like a pipeline service, security policy-as-code configurations defined, and progressive delivery built-in. All an application developer needs to do is click on the template, fill in a few fields, and at the click of a button all the required YAML is automatically produced by the template and ready to be deployed.



GITOPS TEMPLATES
That embed use-case focused stacks, i.e. observability, network policies, etc.

POLICIES
Sets of policies to enforce standards, secure configurations and best practices

WELL-ARCHITECTURE PATTERN
Well-architected patterns; reliability, security, efficiency and operational excellence

“ Leveraging GitOps has allowed us to create a self-service platform for engineers so they can concentrate on delivering business value through innovation, without the full need for Platform Team assistance. The engineers focus on building container images and managing the testing of their microservices and Weave Flux (Weave GitOps) handles the deployments.”

— Steve Wade, Platform Lead, Mettle

READ THE CASE STUDY

Weave GitOps Enterprise Features

Weave GitOps Enterprise boasts a number of other distinct features that empower platform teams to build self-service platforms:

Feature	Platform Team Role	Developer Benefits
Workspaces & Policy as Code Safely manage multi-tenant environments.	▶ Create safe perimeters between users and applications as they share resources via workspaces. ▶ Define and govern policy configurations.	▶ Have the freedom and autonomy to use preassigned resources without worrying about affecting other user's performance and security. ▶ No need for approvals and unnecessary back and forth with Ops teams.
Terraform Controller Manage Terraform resources, the GitOps way.	▶ Create HCL templates & merge with a Kubernetes custom resource to produce valid Terraform HCL code. ▶ Create an end-to-end flow for Terraform users & infrastructure. ▶ Manage cloud infrastructure beyond Kubernetes.	▶ Can use a self-service template to set-up a new bucket from the existing cloud provider. ▶ Reuse these templates to deploy multiple instances with different configurations.
Pipelines Automate release pipelines.	▶ Can enforce automated gates and checks to any deployment and if these checks fail, the deployment is automatically rolled back. ▶ Have complete visibility into how code is progressing through different environments.	▶ Automate the rollout of code from one environment to another without human intervention & with end-to-end visibility. ▶ Can define which Helm charts are part of the environments they create & see which versions of their applications are running in which environment from a single screen.
GitOps Run Automate environment provisioning and tie in security guardrails.	▶ Create & manage developer environments at scale. ▶ Can tweak the policies governing environment creation as easily as editing a YAML file.	▶ Can create consistent developer environments on-demand so they can spend more time doing what they do best - write code.

Feature	Platform Team Role	Developer Benefits
GitOps Templates Pre-built, secure and certified infrastructure templates.	▶ Creates readymade templates for new microservices that would be used by application developers ▶ Simplify the deployment process for developers while having to write little or no YAML. ▶ Ensure requirements are built-in such as pipeline service, policy configuration, and progressive delivery controls.	▶ Easily pick a template, fill in a few fields, and all the required YAML is automatically produced by the template and ready to be deployed. ▶ Can merge pull requests in Git, and the new microservice is deployed to production, complete with all the defaults, and prerequisites as defined by the platform operator.
GitOps Sets A single definition to drive many resources across a fleet of clusters.	▶ Create a single definition of all the objects that are required to successfully deploy an application. ▶ A definition can be used to generate the environment and cluster-specific configuration. ▶ Bootstrap the automation objects for a set of environments.	▶ The platform teams can set up pull request previews for the application teams. This means all new code can deploy to a dev cluster for review by code reviewers and QA. Once the pull request merges the preview is removed from the cluster. ▶ It's now much simpler to configure multiple environments.

The DevOps movement is about removing silos between dev and ops teams. The platform model meets developer and operation teams right where they are, enabling them to do what they do best. As organizations look to move faster, deploy more frequently and become more innovative, the platform model is set to become the norm. Large organizations like State Farm and the DoD are trailblazers; it is time for mainstream organizations to follow in their footsteps and reap the benefits of a self-service developer platform. Weave GitOps Enterprise and its built in GitOps principles are a proven approach for successfully deploying and maintaining a Kubernetes platform that benefits both developers and operators.

CONTACT US FOR A WEAVE GITOPS ENTERPRISE DEMO

sales@weave.works

weave.works